Sections

SECTIONS ...................................................................................................................... II
VERSION HISTORY ....................................................................................................... III
TABLE OF CONTENTS ................................................................................................ VII
TABLE OF FIGURES ..................................................................................................... XVIII
TABLE OF TABLES ....................................................................................................... XIX
GENERAL GUIDELINES .............................................................................................. 1
OPERATIONAL CONCEPT DESCRIPTION (OCD) ....................................................... 6
SYSTEM AND SOFTWARE REQUIREMENTS DEFINITION (SSRD) .............................. 6
SYSTEM AND SOFTWARE ARCHITECTURE DESCRIPTION (SSAD) ......................... 6
LIFE CYCLE PLAN (LCP) ............................................................................................ 6
FEASIBILITY RATIONALE DESCRIPTION (FRD) ....................................................... 6
CONSTRUCTION, TRANSITION, & SUPPORT (CTS) .................................................. 6
ITERATION PLAN ......................................................................................................... 6
ITERATION ASSESSMENT REPORT ........................................................................... 6
RELEASE DESCRIPTION ......................................................................................... 6
QUALITY MANAGEMENT PLAN (QMP) ...................................................................... 6
TEST PLAN ................................................................................................................. 6
TEST DESCRIPTION AND RESULTS ........................................................................... 6
PEER REVIEW PLAN ................................................................................................... 6
PEER REVIEW REPORT .............................................................................................. 6
TRANSITION PLAN ..................................................................................................... 6
SOFTWARE USER'S MANUAL .................................................................................... 6
SYSTEM AND SOFTWARE SUPPORT PLAN (SSSP) .................................................. 6
SOURCES AND REFERENCES ..................................................................................... 6
APPENDICES ............................................................................................................... 6
APPENDIX A. SUGGESTED WIN–WIN TAXONOMY FOR MBASE ............................... 6
APPENDIX B. LEVEL OF SERVICE REQUIREMENTS ................................................ 6
APPENDIX C. COMMON DEFINITION LANGUAGE (CDL) FOR MBASE .................... 6
APPENDIX D. STATUS REPORTS ................................................................................ 6
INDEX .......................................................................................................................... 6
## Guidelines for MBASE

### Version History

<table>
<thead>
<tr>
<th>Date</th>
<th>Author</th>
<th>Version</th>
<th>Changes made</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/31/00</td>
<td>Nikunj Mehta</td>
<td>1.1.0</td>
<td>• Added some details to the LCP Approach section</td>
</tr>
<tr>
<td>2/3/00</td>
<td>Dan Port</td>
<td>1.2.0</td>
<td>• Elaborated and re-wrote parts of Invariants/Variants section</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added model classification per section guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added model classifications to OCD, SSRD, SSAD, LCP, FRD outlines</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Elaborated on Architectural Views SSAD 3.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added details to LCP</td>
</tr>
<tr>
<td>2/14/00</td>
<td>Barry Boehm</td>
<td>1.3.0</td>
<td>• Elaborated several LCP sections</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Re-organized several LCP sections</td>
</tr>
<tr>
<td>2/18/00</td>
<td>Barry Boehm</td>
<td>1.3.1</td>
<td>• Further elaborations to LCP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• BRA material added to LCP 4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Need to re-sync LCP sections with outline TOC</td>
</tr>
<tr>
<td>2/22/00</td>
<td>Ebru Dincel</td>
<td>1.4.0</td>
<td>• Synchronization of Section numbers with the ones in TOC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Reference check</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Took out course related material</td>
</tr>
<tr>
<td>2/23/00</td>
<td>Ebru Dincel</td>
<td>1.4.1</td>
<td>• Added some abbreviations</td>
</tr>
<tr>
<td>2/24/00</td>
<td>Ebru Dincel</td>
<td>1.4.2</td>
<td>• Corrected some typos</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Appendix A of MBASE moved to OCD 5</td>
</tr>
<tr>
<td>2/26/00</td>
<td>Nikunj Mehta</td>
<td>1.4.3</td>
<td>• Edits and corrections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Changed link section under LCP Approach to bullets</td>
</tr>
<tr>
<td>2/29/00</td>
<td>Ebru Dincel</td>
<td>1.4.5</td>
<td>• Edits and corrections to LCP</td>
</tr>
<tr>
<td>7/10/00</td>
<td>Dan Port</td>
<td>1.5</td>
<td>• Updated OCD, added CTS</td>
</tr>
<tr>
<td>8/1/00</td>
<td>Dan Port</td>
<td>1.5.1</td>
<td>• Moved material after FRD to after CTS</td>
</tr>
<tr>
<td>8/7/00</td>
<td>Ebru Dincel</td>
<td>1.5.2</td>
<td>• LCP Rework, split participants into agents</td>
</tr>
<tr>
<td>8/14/00</td>
<td>Dan Port</td>
<td>1.6</td>
<td>• Updated OCD, reworked SSRD (removed User Classes, moved Typical Levels of Service table to OCD), reworked SSAD (sample specification templates, object model)</td>
</tr>
<tr>
<td>8/20/00</td>
<td>Ebru Dincel</td>
<td>1.7</td>
<td>• New OCD 2.0</td>
</tr>
<tr>
<td>8/28/00</td>
<td>Dan Port</td>
<td>1.7.1</td>
<td>• Lots of OCD and SSRD revisions, added Degree of Detail and Tailoring sections</td>
</tr>
<tr>
<td>8/31/00</td>
<td>Dan Port</td>
<td>1.8</td>
<td>• More SSRD revisions, LCP and FRD revisions</td>
</tr>
<tr>
<td>9/4/00</td>
<td>Dan Port</td>
<td>1.9</td>
<td>• Added Course Guidelines to all sections</td>
</tr>
<tr>
<td>9/5/00</td>
<td>Ebru Dincel</td>
<td>2.0</td>
<td>• RUP related material</td>
</tr>
<tr>
<td>11/19/00</td>
<td>A W Brown</td>
<td>2.0a</td>
<td>• SSRD RUP related material repaired based on LCO ARB</td>
</tr>
<tr>
<td>11/24/00</td>
<td>A W Brown</td>
<td>2.0c</td>
<td>• Rest of RUP related material repaired based on LCO ARB</td>
</tr>
<tr>
<td>Date</td>
<td>Author</td>
<td>Version</td>
<td>Changes made</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>12/24/00</td>
<td>A W Brown</td>
<td>2.0d</td>
<td>• RUP related material based on 2000 LCA ARB</td>
</tr>
<tr>
<td>12/10/00</td>
<td>Ebru Dincel</td>
<td>2.1</td>
<td>• Corrected some reference errors in SSRD</td>
</tr>
<tr>
<td>1/6/01</td>
<td>A W Brown</td>
<td>2.1.1</td>
<td>• 2.0d merged with 2.1</td>
</tr>
<tr>
<td>1/25/01</td>
<td>Dan Port</td>
<td>2.2</td>
<td>• Added revised Support Plan (SSSP), clarifications to many sections based on CS577a 2000 ARB reviews, COTS Integration special emphasis, expanded introduction to prototypes OCD 5</td>
</tr>
<tr>
<td>9/2/01</td>
<td>Edward Colbert</td>
<td>2.3</td>
<td>• Revised RUP Guidelines in SSAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Renamed following components of SSAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Design Component Specification as Component Implementation Design Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Logical Component View as Logical Class Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Object Static Structure Model as Object Structure Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Operations Model as Interaction Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Classification Model as Implementation Class Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised SSAD 3.1.2 Component Implementation Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised SSAD 3.5 Configuration Model and added RUP Guidelines.</td>
</tr>
<tr>
<td>9/6/01</td>
<td>Dan Port</td>
<td>2.3.1</td>
<td>• Removed introduction material (now in separate document)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added Jung-Won's new weekly effort form in appendix</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed COTS integration spiral from 3.1.2 Component Implementation Design Model (reference to new supplement added)</td>
</tr>
<tr>
<td>9/26/01</td>
<td>Ed Colbert</td>
<td>2.3.2</td>
<td>• In SSAD,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Minor revisions to package naming convention in RUP Guidelines in sections 2.1, 3.1.1, and 3.1.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Replaced RUP Guideline in section 2.3 with separate guidelines in sections 2.3.1 and 2.3.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added RUP Guideline to section 3.1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed “LCA” from headers of RUP Guidelines</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added LCO–, LCA–, and IOC–specific guides to RUP Guidelines.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• In SSRD,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised section 3.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Remove subsection headers &amp; merged the discussion that was in those subsections into the discussion for section 3.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed RUP Guidelines for creating a use–case model.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed RUP Guidelines for creating a use–case model from section 6.1.</td>
</tr>
<tr>
<td>Date</td>
<td>Author(s)</td>
<td>Version</td>
<td>Changes made</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10/5/01</td>
<td>Ed Colbert</td>
<td>2.3.3</td>
<td>• In OCD,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added 577a Guidelines to Section 2.1.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added RUP Guidelines to Section 2.3.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised RUP Guidelines in Sections 3.4, 3.5, 4.3, 4.5, and 4.5.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• In SSRD,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed RUP Guidelines for creating a use-case model from section 3.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Removed RUP Guidelines for creating a use-case model from section 6.1.</td>
</tr>
<tr>
<td>01/13/02</td>
<td>Ed Colbert and A W Brown</td>
<td>2.3.4</td>
<td>• Minor edits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Change Control Summary section, such as 1.3 for OCD added for 577 to each document with “references”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added Section 1.1 and 1.2 to Test Plan for consistencies</td>
</tr>
<tr>
<td>01/14/02</td>
<td>A W Brown</td>
<td>2.3.5</td>
<td>• Generalized &quot;Inspections&quot; to COQUALMO's &quot;Peer Reviews&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Growing &quot;plans&quot; tree with versions required earlier</td>
</tr>
<tr>
<td>01/31/02</td>
<td>A W Brown</td>
<td>2.3.6b</td>
<td>• Color code changes of 2.3.4 and 2.3.5 in dark red (changes made) or orange (changes [deletions?] to be made).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Today’s changes color coded in RED; to be changed to day in brown.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Significant changes in LCP section 4, Quality Management Plan, Peer Review Plan, Peer Review Report, and Weekly status report.</td>
</tr>
<tr>
<td>01/31/02</td>
<td>Ed Colbert</td>
<td>2.3.6b</td>
<td>• Revised RUP Guideline in SSAD Section 3.5.</td>
</tr>
<tr>
<td>02/02/02</td>
<td>A W Brown</td>
<td>2.3.6c</td>
<td>• Fixed “Change Summary” § heading and accompanying reference to last ARB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Minor additional corrections in QMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Further additions to Peer Review Plan and Peer Review Report</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Clarifications to the weekly status report</td>
</tr>
<tr>
<td>02/14/02</td>
<td>Barry Boehm &amp; Dan Port</td>
<td>2.3.6d</td>
<td>• Minor correction of Transition Plan and QMP</td>
</tr>
<tr>
<td>09/13/02</td>
<td>Ed Colbert, Murali G. &amp; Barry Boehm</td>
<td>2.3.7</td>
<td>• Added Table of Contents to front of document</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added a Table of Figures and a Table of Tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Added Index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Standardized formatting of subdocuments.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Standardized sections in subdocuments</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Edited OCD Sections A, B.1, and B.2; FRD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Minor editorial changes throughout</td>
</tr>
<tr>
<td>Date</td>
<td>Author</td>
<td>Version</td>
<td>Changes made</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>02/11/03</td>
<td>Ed Colbert</td>
<td>2.4</td>
<td>• Revised OCD and SSAD models and structure</td>
</tr>
<tr>
<td></td>
<td>A. W. Brown with Meghna &amp;</td>
<td>2.4</td>
<td>• in LCP</td>
</tr>
<tr>
<td></td>
<td>Sachin Shah</td>
<td></td>
<td>• Incorporated the role of a Quality focal point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised section 3.2 to include testing responsibilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated Section 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Corrected the Key Products &amp; Technical managementReviews table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Updated the Sec 4.3 Configuration management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• and added the concept of weak, informal &amp; strong baselines</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Changed Sec 4.4.1 to Product Assurance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Replaced the Section 4.4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Corrected some reference errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• in QMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised Sections 1.2,2,2,3,2.4,2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised Section 2.6 Testing Guidelines</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• in Peer Review Plan</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Revised Section 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Section 7 Added the snapshots of the updated Agile Internal Review form &amp; description.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In CDL: Added some definitions</td>
</tr>
<tr>
<td></td>
<td>A. W. Brown</td>
<td></td>
<td>• Made status reports after CDL their own appendix. Changed space before/after TOC1 in &quot;Sections&quot; so it fits on one page. Changed page numbers in front matter to lower case roman numerals. Minor editorial changes in LCP, QMP and Test Plan.</td>
</tr>
<tr>
<td></td>
<td>A. W. Brown, M&amp;S Shah</td>
<td></td>
<td>• Product and Process Assurance sub-sections of LCP and QMP improved. More editorial changes in QMP.</td>
</tr>
<tr>
<td></td>
<td>A. W. Brown, Meghna Shah</td>
<td></td>
<td>Added section 1.0 Purpose to the QMP, Peer Review Plan &amp; Peer Review Report for Overview, References &amp; Change Summary information</td>
</tr>
</tbody>
</table>
Table of Contents

SECTIONS .......................................................................................................................... II

VERSION HISTORY ......................................................................................................... III

TABLE OF CONTENTS ..................................................................................................... VII

TABLE OF FIGURES ....................................................................................................... XVIII

TABLE OF TABLES ........................................................................................................ XIX

GENERAL GUIDELINES ............................................................................................... 1

A. MBASE ....................................................................................................................... 1

B. General Formatting Guidelines ................................................................................ 4

C. Final Remark ............................................................................................................. 4

D. Conventions Used ..................................................................................................... 4

OPERATIONAL CONCEPT DESCRIPTION (OCD) ....................................................... 6

A. Description ............................................................................................................... 6

1. Purpose ..................................................................................................................... 6

2. Completion Criteria .................................................................................................. 6

   2.1 Life Cycle Objectives (LCO) ............................................................................. 6

   2.2 Life Cycle Architecture (LCA) ........................................................................ 6

   2.3 Initial Operational Capability (IOC) ................................................................. 6

3. Intended Audience ................................................................................................... 6

4. Participating Agents ................................................................................................. 6

5. Performing Agents .................................................................................................... 6

6. High-Level Dependencies ........................................................................................ 6

7. Degree of Detail and Tailoring ................................................................................ 6

B. Document Sections .................................................................................................. 6

1. Introduction .............................................................................................................. 6

   1.1 Purpose of the OCD Document ....................................................................... 6

   1.2 References .......................................................................................................... 6

   1.3 Change Summary ............................................................................................... 6

2. Shared Vision ............................................................................................................ 6

   2.1 System Capability Description ......................................................................... 6

   2.2 Key Stakeholders ............................................................................................... 6

   2.3 System Boundary and Environment ............................................................... 6

   2.4 Major Project Constraints ................................................................................ 6

   2.5 Top-Level Business Case ................................................................................ 6

   2.6 Inception Phase Plan and Required Resources ............................................ 6

   2.7 Initial Spiral Objectives, Constraints, Alternatives, and Risks ..................... 6

3. Domain/Organization Description ........................................................................... 6

   3.1 Organization Background ................................................................................ 6
### Table of Contents

1. Introduction ........................................................................................................... 6
2. Project Requirements........................................................................................................ 6
   2.1 Life Cycle Objectives (LCO) .................................................................................. 6
   2.2 Life Cycle Architecture (LCA) ............................................................................... 6
   2.3 Initial Operational Capability (IOC) ......................................................................... 6
3. Intended audience ............................................................................................................ 6
4. Participating Agent ......................................................................................................... 6
5. Performing Agent ............................................................................................................ 6
6. High-Level Dependencies .............................................................................................. 6
7. Degree of Detail and Tailoring ....................................................................................... 6

### SYSTEM AND SOFTWARE REQUIREMENTS DEFINITION (SSRD) .......................... 6

#### A. Description .............................................................................................................. 6
1. Purpose ................................................................................................................... 6
2. Completion Criteria........................................................................................................ 6
   2.1 Life Cycle Objectives (LCO) .................................................................................. 6
   2.2 Life Cycle Architecture (LCA) ............................................................................... 6
   2.3 Initial Operational Capability (IOC) ......................................................................... 6
3. Intended audience ......................................................................................................... 6
4. Participating Agent ....................................................................................................... 6
5. Performing Agent .......................................................................................................... 6
6. High-Level Dependencies ........................................................................................... 6
7. Degree of Detail and Tailoring ....................................................................................... 6

#### B. Document Sections ................................................................................................ 6
1. Introduction .................................................................................................................. 6
   1.1 Purpose of the System and Software Requirements Definition Document.............. 6
   1.2 References ............................................................................................................... 6
   1.3 Change Summary .................................................................................................... 6
2. Project Requirements .................................................................................................... 6
   2.1 Budget and Schedule .............................................................................................. 6
   2.2 Development Requirements .................................................................................. 6
   2.3 Deployment Requirements .................................................................................... 6
   2.4 Transition Requirements ....................................................................................... 6
   2.5 Support Environment Requirements ....................................................................... 6
3. Capability Requirements ............................................................................................... 6
   3.1 System Definition .................................................................................................. 6
   3.2 System Requirements ........................................................................................... 6
4. System Interface Requirements ..................................................................................... 6
   4.1 User Interface Standards Requirements .................................................................. 6
   4.2 Hardware Interface Requirements ......................................................................... 6
   4.3 Communications Interface Requirements .............................................................. 6
   4.4 Other Software Interface Requirements ............................................................... 6
5. Level of Service (L.O.S.) Requirements ......................................................................... 6
6. Evolution Requirements ................................................................................................ 6
   6.1 Capability Evolution Requirements ....................................................................... 6
   6.2 Interface Evolution Requirements ......................................................................... 6
   6.3 Technology Evolution Requirements ...................................................................... 6
SYSTEM AND SOFTWARE ARCHITECTURE DESCRIPTION (SSAD) ...................... 6

A. Description .......................................................................................................................... 6
   1. Purpose .................................................................................................................................. 6
   2. Completion Criteria .............................................................................................................. 6
   2.1 Life Cycle Objectives (LCO) .......................................................................................... 6
   2.2 Life Cycle Architecture (LCA) ......................................................................................... 6
   2.3 Initial Operational Capability (IOC) ............................................................................... 6
   3. Intended audience .................................................................................................................. 6
   4. Participating Agent .............................................................................................................. 6
   5. Performing Agent ................................................................................................................ 6
   6. High-Level Dependencies ................................................................................................... 6
   7. Overall Tool Support .......................................................................................................... 6
   8. Degree of Detail and Tailoring ............................................................................................. 6

B. Document Sections .............................................................................................................. 6
   1. Introduction .......................................................................................................................... 6
      1.1 Purpose of the SSAD Document ................................................................................... 6
      1.2 Standards and Conventions ......................................................................................... 6
      1.3 References .................................................................................................................... 6
      1.4 Change Summary .......................................................................................................... 6
   2. System Analysis ................................................................................................................... 6
      2.1 Structure ........................................................................................................................ 6
      2.2 Artifacts & Information ............................................................................................... 6
      2.3 Behavior ....................................................................................................................... 6
      2.4 L.O.S. Goals .................................................................................................................. 6
      2.5 Rules .............................................................................................................................. 6
   3. Architecture Design & Analysis .......................................................................................... 6
      3.1 Structure ........................................................................................................................ 6
      3.2 Analysis Classes ............................................................................................................ 6
      3.3 Behavior ....................................................................................................................... 6
      3.4 L.O.S., Projected ........................................................................................................... 6
      3.5 Architectural Styles, Patterns & Frameworks ................................................................. 6
   4. Implementation Design ....................................................................................................... 6
      4.1 Structure ........................................................................................................................ 6
      4.2 Behavior ....................................................................................................................... 6
      4.3 L.O.S. ............................................................................................................................ 6
      4.4 Patterns & Frameworks .................................................................................................. 6
      4.5 Project Artifacts ............................................................................................................ 6
   5. Glossary for System Analysis and Design ........................................................................... 6
   6. Appendices .......................................................................................................................... 6

LIFE CYCLE PLAN (LCP) ...................................................................................................... 6

A. Description .......................................................................................................................... 6
   1. Purpose .................................................................................................................................. 6
   2. Completion Criteria .............................................................................................................. 6
   2.1 Life Cycle Objectives (LCO) .......................................................................................... 6
   2.2 Life Cycle Architecture (LCA) ......................................................................................... 6

MBASE_Guidelines_v2.4.0.doc ix Version Date: 2/11/2003
### Guidelines for MBASE

<table>
<thead>
<tr>
<th>FEASIBILITY RATIONALE DESCRIPTION (FRD)</th>
<th>6</th>
</tr>
</thead>
</table>

#### A. Description

1. Purpose .......................................................... 6
2. Completion Criteria ........................................... 6
   2.1 Life Cycle Objectives (LCO) ......................... 6
   2.2 Life Cycle Architecture (LCA) .................. 6
   2.3 Initial Operational Capability (IOC) ........ 6
3. Intended Audience ............................................. 6
4. Participating Agent ............................................ 6
5. Performing Agent ............................................... 6
6. High-Level Dependencies .................................... 6
7. Overall Tool Support ......................................... 6
8. Degree of Detail and Tailoring ......................... 6

#### B. Document Sections

1. Introduction ...................................................... 6
   1.1 Purpose of the Feasibility Rationale Description Document 6
   1.2 References .................................................... 6
   1.3 Change Summary ........................................... 6
2. Product Rationale ................................................. 6
   2.1 Business Case Analysis .................................. 6
   2.2 Requirements Satisfaction .............................. 6
   2.3 Stakeholder Concurrence .............................. 6
Guidelines for MBASE

Table of Contents

3. Process Rationale ............................................................................................................................................. 6
   3.1 System Priorities .......................................................................................................................................... 6
   3.2 Process Match to System Characteristics and Priorities ............................................................................. 6
   3.3 Consistency of Priorities, Process and Resources ....................................................................................... 6
4. Project Risk Assessment .................................................................................................................................... 6
5. Analysis Results .................................................................................................................................................. 6
   5.1 Product Features .......................................................................................................................................... 6
   5.2 Commercial-Off-The-Shelf Solutions .......................................................................................................... 6
6. Appendices ....................................................................................................................................................... 6

CONSTRUCTION, TRANSITION, & SUPPORT (CTS) ................................................................. 6

A. General Construction Process Guidelines ...................................................................................................... 6
B. Guidelines for the Deliverables .......................................................................................................................... 6
C. General Guidelines for Plans and Reports ........................................................................................................... 6

ITERATION PLAN ........................................................................................................................................... 6

A. Description ....................................................................................................................................................... 6
   1. Purpose ............................................................................................................................................................ 6
   2. Timing ............................................................................................................................................................. 6
   3. Intended Audience ....................................................................................................................................... 6
   4. Participants .................................................................................................................................................... 6
   5. Responsibility ............................................................................................................................................... 6
   6. Completion Criteria ..................................................................................................................................... 6
   7. Additional Information ................................................................................................................................. 6
   8. High-Level Dependencies ............................................................................................................................ 6

B. Document Sections .......................................................................................................................................... 6
   1. Iteration Overview ....................................................................................................................................... 6
      1.1 Capabilities to be Implemented ............................................................................................................... 6
      1.2 Capabilities to be Tested ........................................................................................................................... 6
      1.3 Capabilities not to be tested ....................................................................................................................... 6
      1.4 Objectives ............................................................................................................................................... 6
   2. Plan ............................................................................................................................................................... 6
      2.1 Tool Support ............................................................................................................................................ 6
      2.2 Schedule of Activities ............................................................................................................................... 6
      2.3 Resources ............................................................................................................................................... 6
      2.4 Team Responsibilities ............................................................................................................................... 6
   3. Approach ....................................................................................................................................................... 6
   4. Assumptions .................................................................................................................................................. 6

ITERATION ASSESSMENT REPORT ................................................................................................. 6

A. Description ....................................................................................................................................................... 6
   1. Purpose ............................................................................................................................................................ 6
   2. Timing ............................................................................................................................................................. 6
   3. Intended Audience ....................................................................................................................................... 6
   4. Participants .................................................................................................................................................... 6
   5. Responsibility ............................................................................................................................................... 6
   6. Completion Criteria ..................................................................................................................................... 6
# Guidelines for MBASE

<table>
<thead>
<tr>
<th>Document Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Description</td>
<td>6</td>
</tr>
<tr>
<td>1. Purpose</td>
<td>6</td>
</tr>
<tr>
<td>2. Timing</td>
<td>6</td>
</tr>
<tr>
<td>3. Intended Audience</td>
<td>6</td>
</tr>
<tr>
<td>4. Participants</td>
<td>6</td>
</tr>
<tr>
<td>5. Responsibility</td>
<td>6</td>
</tr>
<tr>
<td>6. Completion Criteria</td>
<td>6</td>
</tr>
<tr>
<td>7. Additional Information</td>
<td>6</td>
</tr>
<tr>
<td>8. High-Level Dependencies</td>
<td>6</td>
</tr>
<tr>
<td>B. Document Sections</td>
<td>6</td>
</tr>
<tr>
<td>1. Overview</td>
<td>6</td>
</tr>
<tr>
<td>1.1 Capabilities Implemented</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Summary of Test Results</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Open Problems</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Objectives Reached</td>
<td>6</td>
</tr>
<tr>
<td>2. Adherence to Plan</td>
<td>6</td>
</tr>
<tr>
<td>3. Approach Used and Suggested Changes</td>
<td>6</td>
</tr>
<tr>
<td>4. External Changes Occurred</td>
<td>6</td>
</tr>
<tr>
<td>5. Suggested Actions</td>
<td>6</td>
</tr>
<tr>
<td>6. Defect and Change Request Reporting Procedures</td>
<td>6</td>
</tr>
<tr>
<td>7. Additional Information</td>
<td>6</td>
</tr>
<tr>
<td>8. High-Level Dependencies</td>
<td>6</td>
</tr>
<tr>
<td>9. Quality Management Plan (QMP)</td>
<td>6</td>
</tr>
<tr>
<td>A. Description</td>
<td>6</td>
</tr>
<tr>
<td>1. Purpose</td>
<td>6</td>
</tr>
<tr>
<td>2. Timing</td>
<td>6</td>
</tr>
<tr>
<td>3. Intended Audience</td>
<td>6</td>
</tr>
<tr>
<td>4. Participants</td>
<td>6</td>
</tr>
<tr>
<td>5. Responsibility</td>
<td>6</td>
</tr>
<tr>
<td>6. Completion Criteria</td>
<td>6</td>
</tr>
<tr>
<td>7. Additional Information</td>
<td>6</td>
</tr>
<tr>
<td>8. High-Level Dependencies</td>
<td>6</td>
</tr>
</tbody>
</table>

**Release Description**

**Quality Management Plan (QMP)**
Guidelines for MBASE

Table of Contents

1. Purpose ..............................................................................................................................................................6
1.1 Overview ..........................................................................................................................................................6
1.2 References .....................................................................................................................................................6
1.3 Change Summary ...........................................................................................................................................6
2. Quality Guidelines ...............................................................................................................................................6
2.1 Design Guidelines ......................................................................................................................................6
2.2 Coding Guidelines ......................................................................................................................................6
2.3 Quality Assessment Methods .......................................................................................................................6
2.4 Process Assurance ......................................................................................................................................6
2.5 Product Assurance ......................................................................................................................................6
2.6 Problem Reporting and Tracking System .....................................................................................................6
2.7 Configuration Management ..........................................................................................................................6
2.8 Testing Guidelines ......................................................................................................................................6
2.9 Defect and Change Management ................................................................................................................6
3. Appendix ..........................................................................................................................................................6
3.1 Appendix with Problem Reporting Form/System ........................................................................................6
3.2 Appendix with Agile Artifact Review Form Set and Instructions ................................................................6
3.3 Appendix with Fagan’s Inspection Form Set and Instructions .....................................................................6

TEST PLAN ........................................................................................................................................................... 6

A. Description ..........................................................................................................................................................6
1. Purpose ..............................................................................................................................................................6
2. Timing .................................................................................................................................................................6
3. Intended Audience ........................................................................................................................................6
4. Participants ........................................................................................................................................................6
5. Responsibility ....................................................................................................................................................6
6. Completion Criteria ........................................................................................................................................6
7. Additional Information ..................................................................................................................................6
8. High-Level Dependencies ..............................................................................................................................6

B. Document Section ........................................................................................................................................6
1. Introduction .......................................................................................................................................................6
1.1 Purpose of the Test Plan ................................................................................................................................6
1.2 References .....................................................................................................................................................6
1.3 Change Summary ........................................................................................................................................6
2. Environment Preparation ................................................................................................................................6
2.1 Hardware preparation ................................................................................................................................6
2.2 Software preparation ................................................................................................................................6
2.3 Other pre-test preparations ..........................................................................................................................6
3. Test Identification ...........................................................................................................................................6
3.1 Test Identifier ................................................................................................................................................6
4. Resources ..........................................................................................................................................................6
4.1 Responsibilities ............................................................................................................................................6
4.2 Staffing and Training Needs ........................................................................................................................6
4.3 Schedule for Testing Activities ....................................................................................................................6
4.4 Other resource allocations ...........................................................................................................................6
5. Test Completion Criteria ................................................................................................................................6

TEST DESCRIPTION AND RESULTS .................................................................................................................. 6

A. Description ..........................................................................................................................................................6
1. Purpose ..............................................................................................................................................................6
2. Timing .................................................................................................................................................................6
Guidelines for MBASE Table of Contents

B. Document Sections .................................................................................................................. 6

1. Introduction ........................................................................................................................... 6
   1.1 Purpose ............................................................................................................................ 6
   1.2 References ....................................................................................................................... 6
   1.3 Change Summary ............................................................................................................ 6
2. Test Identification .................................................................................................................... 6
3. Test Preparation ...................................................................................................................... 6
   3.1 Test Identifier ................................................................................................................ 6
4. Test Case Description ............................................................................................................. 6
   4.1 Test Identifier ................................................................................................................ 6
5. Test Incident Reports ............................................................................................................. 6
   5.1 Identifier ....................................................................................................................... 6
   5.2 Summary ...................................................................................................................... 6
   5.3 Incident Description ..................................................................................................... 6
   5.4 Impact ........................................................................................................................... 6
6. Test Log .................................................................................................................................. 6
   6.1 Test-Log Identifier ........................................................................................................ 6
   6.2 Test Record .................................................................................................................... 6
7. Test Summary ....................................................................................................................... 6
   7.1 Overall assessment of the software tested: ................................................................. 6
   7.2 Impact of test environment: ....................................................................................... 6
   7.3 Recommended improvements: .................................................................................... 6
8. Notes ...................................................................................................................................... 6
9. Appendices ............................................................................................................................ 6
  Appendix 1: Test log .............................................................................................................. 6

PEER REVIEW PLAN ............................................................................................................... 6

A. Description ........................................................................................................................... 6

1. Purpose .................................................................................................................................. 6
2. Timing .................................................................................................................................... 6
3. Intended Audience ............................................................................................................... 6
4. Participants ........................................................................................................................... 6
5. Responsibility ....................................................................................................................... 6
6. Completion Criteria ............................................................................................................. 6
7. Additional Information ....................................................................................................... 6
8. High-Level Dependencies .................................................................................................. 6

B. Document Sections .............................................................................................................. 6

1. Purpose .................................................................................................................................. 6
   1.1 Overview ........................................................................................................................ 6
   1.2 Since there can be both team and independent (IV&Ver) in-process, people-based reviews for CS577, both should be described if you have an IV&Ver assigned to your project. For sections 2 through 4, this can be done using two separate sub-sections. ............................................................... 6
   1.3 References .................................................................................................................... 6
   1.4 Change Summary .......................................................................................................... 6
2. Peer Review Items, Participants and Roles ......................................................................... 6
3. Peer Review Milestones ...................................................................................................... 6
Guidelines for MBASE  

Table of Contents

4. Peer Review Process ........................................................................................................... 6
5. Classification of Defects ....................................................................................................... 6
   5.1 Severity .......................................................................................................................... 6
   5.2 Type ............................................................................................................................. 6
   5.3 Category ....................................................................................................................... 6
6. Schedule for Peer Reviews ................................................................................................. 6
7. Appendices: Forms for Peer Review Data Gathering .......................................................... 6
   Appendix 2: Fagan's Inspection Data Reporting Forms ......................................................... 6
7.1 Agile Data Reporting Forms ............................................................................................. 6

PEER REVIEW REPORT ........................................................................................................... 6

A. Description ......................................................................................................................... 6
   1. Purpose ............................................................................................................................ 6
   2. Timing ............................................................................................................................. 6
   3. Intended Audience ......................................................................................................... 6
   4. Participants ..................................................................................................................... 6
   5. Responsibility ................................................................................................................ 6
   6. Completion Criteria ....................................................................................................... 6
   7. Additional Information ................................................................................................. 6
   8. High-Level Dependencies ............................................................................................. 6

B. Document Sections ............................................................................................................ 6
   1. Purpose ............................................................................................................................ 6
      1.1 Overview ..................................................................................................................... 6
      1.2 References .................................................................................................................. 6
      1.3 Change Summary ....................................................................................................... 6
   2. Individual Inspection-like Peer Reviews ........................................................................ 6
      2.1 Participants and Roles ............................................................................................... 6
      2.2 Peer Review Items ..................................................................................................... 6
      2.3 Pre-Peer Review Meeting Defect Data ..................................................................... 6
      2.4 Peer Review Meeting Defect Data ........................................................................... 6
      2.5 Post-Peer Review Meeting Defect Data ................................................................... 6
      2.6 Peer Review Statistics ............................................................................................. 6
      2.7 Defect Correction and Rework ................................................................................ 6
   3. Other types of Peer Reviews ........................................................................................... 6
   4. Peer Review Summary Information ................................................................................ 6
      4.1 Peer Reviews' Performed .......................................................................................... 6
      4.2 Peer Review Results ................................................................................................. 6
   5. Appendix ........................................................................................................................ 6

TRANSITION PLAN .................................................................................................................. 6

A. Description ........................................................................................................................... 6
   1. Purpose ............................................................................................................................ 6
   2. Timing ............................................................................................................................. 6
   3. Intended Audience ......................................................................................................... 6
   4. Participants ..................................................................................................................... 6
   5. Responsibility ................................................................................................................ 6
   6. Completion Criteria ....................................................................................................... 6
   7. Additional Information ................................................................................................. 6
   8. High-Level Dependencies ............................................................................................. 6

B. Document Sections ............................................................................................................ 6
**Guidelines for MBASE**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Transition Strategy</td>
<td>6</td>
</tr>
<tr>
<td>1.1 Transition Objectives</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Transition Process Strategy</td>
<td>6</td>
</tr>
<tr>
<td>2. Preparing for Transition</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Hardware Preparation</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Software Preparation</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Site Preparation</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Staff Preparation</td>
<td>6</td>
</tr>
<tr>
<td>2.5 Operational Test and Evaluation</td>
<td>6</td>
</tr>
<tr>
<td>3. Stakeholder Roles and Responsibilities</td>
<td>6</td>
</tr>
<tr>
<td>4. Milestone Plan</td>
<td>6</td>
</tr>
<tr>
<td>5. Required Resources</td>
<td>6</td>
</tr>
<tr>
<td>6. Completion Criteria</td>
<td>6</td>
</tr>
<tr>
<td>7. Additional Information</td>
<td>6</td>
</tr>
<tr>
<td>8. High-Level Dependencies</td>
<td>6</td>
</tr>
</tbody>
</table>

**SOFTWARE USER'S MANUAL**

A. **Description**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Purpose</td>
<td>6</td>
</tr>
<tr>
<td>2. Timing</td>
<td>6</td>
</tr>
<tr>
<td>3. Intended Audience</td>
<td>6</td>
</tr>
<tr>
<td>4. Participants</td>
<td>6</td>
</tr>
<tr>
<td>5. Responsibility</td>
<td>6</td>
</tr>
<tr>
<td>6. Completion Criteria</td>
<td>6</td>
</tr>
<tr>
<td>7. Additional Information</td>
<td>6</td>
</tr>
<tr>
<td>8. High-Level Dependencies</td>
<td>6</td>
</tr>
</tbody>
</table>

**Document Sections**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>6</td>
</tr>
<tr>
<td>1.1 System Overview</td>
<td>6</td>
</tr>
<tr>
<td>1.2 System Requirements</td>
<td>6</td>
</tr>
<tr>
<td>2. Operational Procedures</td>
<td>6</td>
</tr>
<tr>
<td>3. Installation Procedures</td>
<td>6</td>
</tr>
<tr>
<td>3.1 Initialization Procedures</td>
<td>6</td>
</tr>
<tr>
<td>3.2 Re-installation</td>
<td>6</td>
</tr>
<tr>
<td>3.3 De-installation</td>
<td>6</td>
</tr>
<tr>
<td>4. Troubleshooting</td>
<td>6</td>
</tr>
<tr>
<td>4.1 Frequently Asked Questions</td>
<td>6</td>
</tr>
<tr>
<td>4.2 Error Codes and Messages</td>
<td>6</td>
</tr>
<tr>
<td>5. Notes</td>
<td>6</td>
</tr>
<tr>
<td>6. Appendix</td>
<td>6</td>
</tr>
</tbody>
</table>

**SYSTEM AND SOFTWARE SUPPORT PLAN (SSSP)**

A. **Description**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Purpose</td>
<td>6</td>
</tr>
<tr>
<td>2. Timing</td>
<td>6</td>
</tr>
<tr>
<td>3. Intended Audience</td>
<td>6</td>
</tr>
<tr>
<td>4. Participants</td>
<td>6</td>
</tr>
<tr>
<td>5. Responsibilities</td>
<td>6</td>
</tr>
<tr>
<td>6. Completion Criteria</td>
<td>6</td>
</tr>
<tr>
<td>7. Additional Information</td>
<td>6</td>
</tr>
<tr>
<td>8. High-Level Dependencies</td>
<td>6</td>
</tr>
</tbody>
</table>

B. **Document Sections**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Support Objectives and Assumptions</td>
<td>6</td>
</tr>
</tbody>
</table>

**Guidelines for MBASE Table of Contents**

1. Support Objectives and Assumptions.................................................. 6
2. Preparing for Transition....................................................................... 6
  2.1 Hardware Preparation ....................................................................... 6
  2.2 Software Preparation ..................................................................... 6
  2.3 Site Preparation ........................................................................... 6
  2.4 Staff Preparation ......................................................................... 6
  2.5 Operational Test and Evaluation.................................................... 6
3. Stakeholder Roles and Responsibilities.............................................. 6
4. Milestone Plan ..................................................................................... 6
5. Required Resources ............................................................................. 6
6. Completion Criteria ............................................................................ 6
7. Additional Information ....................................................................... 6
8. High-Level Dependencies ..................................................................... 6
### Guidelines for MBASE

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Support Objectives</td>
<td>6</td>
</tr>
<tr>
<td>1.2</td>
<td>Support</td>
<td>6</td>
</tr>
<tr>
<td>2.</td>
<td>Support Strategy and Environment</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>Support Strategy</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Support Environment</td>
<td>6</td>
</tr>
<tr>
<td>3.</td>
<td>Support Responsibilities</td>
<td>6</td>
</tr>
<tr>
<td>4.</td>
<td>Support Approach</td>
<td>6</td>
</tr>
<tr>
<td>5.</td>
<td>Support Resources</td>
<td>6</td>
</tr>
<tr>
<td>5.1</td>
<td>Support Budgets</td>
<td>6</td>
</tr>
<tr>
<td>5.2</td>
<td>Other resources</td>
<td>6</td>
</tr>
</tbody>
</table>

### SOURCES AND REFERENCES

- A. Overall | 6 |
- B. Operational Concept Description | 6 |
- C. System and Software Requirements Definition | 6 |
- D. System and Software Architecture Description | 6 |
- E. Life Cycle Plan | 6 |

### APPENDICES

- APPENDIX A. SUGGESTED WIN–WIN TAXONOMY FOR MBASE | 6 |
- APPENDIX B. LEVEL OF SERVICE REQUIREMENTS | 6 |
- APPENDIX C. COMMON DEFINITION LANGUAGE (CDL) FOR MBASE | 6 |
- APPENDIX D. STATUS REPORTS | 6 |

### INDEX | 6 |
## Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Benefits Realization Approach Results Chain Example</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Context Diagram</td>
<td>6</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Business Use–Case Description</td>
<td>6</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Use–Case Description</td>
<td>6</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Use–Case Realization Description</td>
<td>6</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Operation Description</td>
<td>6</td>
</tr>
<tr>
<td>Figure 7</td>
<td>CS577 Lifecycle Milestones</td>
<td>6</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Sample CS 577b Initial Construction Schedule</td>
<td>6</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Sample schedule for iteration activities within 577b</td>
<td>6</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Area of Concern Log Form</td>
<td>6</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Problems/Issuer Log Cover Sheet Form</td>
<td>6</td>
</tr>
</tbody>
</table>
## Table of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organizational Goal Form</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Project Goal M.A.R.S. Specification Form</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>System Capability Specification</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>Level of Service Specification Form</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Typical Level of Service Concerns of Stakeholders</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>M.A.R.S. Form for a Project Requirement</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>M.A.R.S. Form for Level of a Service Requirement</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>Available Capabilities &amp; Processes in Mode</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>Map of L.O.S. Requirements to Goals for System Processes, Modes, and Information</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>Sample Map of L.O.S. Requirements to Goals for System Processes, Modes, and Information</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>Level of Service Specification Form</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>Map of Organization Rules to System Rules and Elements</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>Parameter Table for Classifiers</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>Component Processes Available in Mode</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>Map of Component Constraints to Sources and Elements</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>Connector Processes Available in Mode</td>
<td>6</td>
</tr>
<tr>
<td>17</td>
<td>Map of Connector Constraints to Sources and Elements</td>
<td>6</td>
</tr>
<tr>
<td>18</td>
<td>L.O.S. Values for a Classifier Instance</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>Architecture L.O.S. Table</td>
<td>6</td>
</tr>
<tr>
<td>20</td>
<td>Architecture Styles, Patterns, and Frameworks</td>
<td>6</td>
</tr>
<tr>
<td>21</td>
<td>Class Attributes</td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>Stereotypes for Project Artifacts</td>
<td>6</td>
</tr>
<tr>
<td>23</td>
<td>Stakeholder responsibilities during the software life cycle</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>Key Products &amp; Technical Management Reviews (Excluding In-Process Peer Reviews)</td>
<td>6</td>
</tr>
<tr>
<td>25</td>
<td>Key Products &amp; Technical Management Reviews (Excluding In-Process Peer Reviews) For CS577ab</td>
<td>6</td>
</tr>
<tr>
<td>26</td>
<td>Top-N Risk Item List (Assuming weekly risk reassessment)</td>
<td>6</td>
</tr>
<tr>
<td>27</td>
<td>Example of Table of Assessment Activities history</td>
<td>6</td>
</tr>
<tr>
<td>28</td>
<td>Miscellaneous Software Project Expenditure Sources</td>
<td>6</td>
</tr>
<tr>
<td>29</td>
<td>Level of Service Product and Process Strategies</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>Level of Service Strategies and Relations: Architecture Strategies</td>
<td>6</td>
</tr>
<tr>
<td>31</td>
<td>Top-Level Field Guide to Software Architecture Attribute Analysis Methods</td>
<td>6</td>
</tr>
<tr>
<td>32</td>
<td>Process Model Decision Table</td>
<td>6</td>
</tr>
<tr>
<td>33</td>
<td>Conditions for Additional Complementary Process Model Options</td>
<td>6</td>
</tr>
<tr>
<td>34</td>
<td>Software Risk Management Techniques</td>
<td>6</td>
</tr>
<tr>
<td>35</td>
<td>Hypothetical Example of Table of Planned Assessment Activities</td>
<td>6</td>
</tr>
<tr>
<td>36</td>
<td>Field Descriptions for the Areas of Concern Log</td>
<td>6</td>
</tr>
<tr>
<td>37</td>
<td>Field Descriptions for Artifact Review Problem List</td>
<td>6</td>
</tr>
</tbody>
</table>
Guidelines for MBASE

Table of Contents

AgileInternalReview_form-fieldDesc(exCeBASE-ToBe-v1.2a) -- Problem List Field Description

This form is to be filled out by the meeting recorder and/or artifact under review's author; author has ultimate responsibility for quality of information.

<table>
<thead>
<tr>
<th>Filed Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Name</td>
<td>Insert the project name or title. For purpose of the class, it should be the team name, such as Team 3.</td>
</tr>
<tr>
<td>Review #</td>
<td>Identify the review number. It is composed of two parts, one to identify the module and the other to identify number of review performed on the module (including version). For example if it is second review of OCD OCD - 2.</td>
</tr>
<tr>
<td>Artifact</td>
<td>Identify the name of the artifact with version number and/or its section numbers being reviewed e.g. OCD or SSRD 2.1 to SSRD 2.3</td>
</tr>
<tr>
<td>Date</td>
<td>Identify the date(s) the list is filled/marked.</td>
</tr>
<tr>
<td>Module</td>
<td>Identify the name of the software/code module (e.g. class etc) and/or the range of the line numbers which are being reviewed.</td>
</tr>
<tr>
<td>Type of review</td>
<td>Identify the type of review like Agile Internal, Agile Artifact etc.</td>
</tr>
<tr>
<td>MBASE/Phase/level</td>
<td>Inception, Elaboration, Construction or Transition. If there are cycles within a phase, then please indicate which cycle as well.</td>
</tr>
<tr>
<td>Review Date(s)</td>
<td>Identify the date(s) that the review was conducted.</td>
</tr>
<tr>
<td>Defects Found</td>
<td>Identify number of Major and Minor defects found. Major defects are those that would result in a failure during operation or otherwise violate a client's requirements.</td>
</tr>
<tr>
<td>No. of Open Issues</td>
<td>Issues are problems that the &quot;Author&quot; can not fix solely in this artifact or at this time.</td>
</tr>
<tr>
<td>No. of Unavoidable defects</td>
<td>Unavoidable defects (aka changes) arise because of the methods, techniques or approaches being followed necessitate changes. Examples include changes arising because of the dynamics of learning, exploration in IOW, situations, code or screen contents reorganization taken on as an &quot;afterthought&quot;, replacement of stubs or placeholders in code, etc. Such situations are often &quot;planned for&quot; and expected to occur.</td>
</tr>
<tr>
<td>No. of Avoidable defects</td>
<td>Changes in analysis, design, code or documentation arising from human error, and which could be avoided through better analysis, design, training, etc. Examples include stub replacement that violates win conditions or requirements such as execution time, memory space, for instance the replacement of a &quot;stub&quot; which breaks a critical timing constraint.</td>
</tr>
<tr>
<td>Comments:</td>
<td>Any Quality assessment issues, or process or improvement suggestions.</td>
</tr>
<tr>
<td>Defects/Issue</td>
<td>An indication of the AUTHOR'S classification of the fixes. &quot;D&quot; for Defects and &quot;I&quot; for issue. Defects are things the AUTHOR has or will fix in the document under review; issues are things that can not be fixed in this artifact or are related to other artifacts which the author can NOT fix directly or can not be fixed at this time, [Used just to ease the counting and avoid losing information.]</td>
</tr>
<tr>
<td>#:</td>
<td>A running number beginning with &quot;D&quot; for Defects and &quot;I&quot; for Issues, used just to ease the counting and avoid losing information.</td>
</tr>
<tr>
<td>Location(s)</td>
<td>If document, identify section, paragraph, and/or line/sentence number of the defect or issue. If program, identify file name, subroutine, and line number.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the defect. Include the PBR &quot;wrong&quot; defect sub-classifications of &quot;Incorrect Fact&quot;, &quot;Inconsistency&quot;, or &quot;Ambiguity&quot;, if appropriate.</td>
</tr>
<tr>
<td>Concern #</td>
<td>Identify the number of concern from the Area of Concern Log to which this Di belongs. This is to map the Defect/Issue reported here (at the Problem List) to the concern that was raised in the &quot;Area of Concern Log&quot;.</td>
</tr>
<tr>
<td>Classification</td>
<td>The classification of the fix (defect) as Major or Minor, or Issue as Open. Major defects are those that would result in a failure during operation or otherwise violate a client's requirements.</td>
</tr>
</tbody>
</table>

© 2002 A.W. Brown, BBSN/See USC CE/IS/IT Review_form-fieldDesc(exCeBASE-ToBe-v1.2a)—Problem List Field Description
1/1/2003
Guidelines for MBASE

General Guidelines

Please read the following general guidelines carefully, before proceeding to the guidelines for the individual deliverables.

A. MBASE

Model-based System Architecting and Software Engineering (MBASE) is an approach that integrates the process (PS), product (PD), property (PY) and success (SS) models for developing a software system. The essence of the approach is to develop the following system definition elements concurrently and iteratively (or by refinement) using the Win–Win Spiral approach defined in [Boehm, 1996].

- Operational Concept Description (OCD)
- System and Software Requirements Definition (SSRD)
- System and Software Architecture Description (SSAD)
- Life Cycle Plan (LCP)
- Feasibility Rationale Description (FRD)
- Construction, Transition, Support (CTS) plans and reports
- Risk-driven prototypes

The three critical project milestones are the Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and the Initial Operating Capability (IOC). The system definition elements have to satisfy specific completion criteria at each anchor point.

- The system definition elements are strongly integrated and a strong traceability thread ties the various sections: e.g., the System Definition (documented in the SSRD) is a refinement of the Statement of Purpose (documented in the OCD). Therefore, to enforce conceptual integrity, it is essential that team members work collaboratively, particularly on strongly related sections.

- Due to the strong interdependencies, it may be a good idea to follow some order when producing the deliverables, at least initially: e.g., write core sections of the OCD before the SSRD. During successive iterations, the documents generally should not be traversed in a linear fashion. Forward consistency should always be enforced (if an Entity is added to the Entity Model, then it should be examined as to how it affects the Component Model). Backward consistency can be less strongly enforced, but is useful to do where feasible.

- Strongly dependent sections are indicated by [Consistent with DDD x.x.x] where DDD is the LCO/LCA deliverable, and x.x.x the section number. When reviewing the deliverables and checking the overall conceptual integrity, it is very helpful to review strongly connected sections in sequence (e.g., OCD: Statement of Purpose, SSRD: System Definition), as opposed to reviewing the deliverables in a linear fashion.

- Conceptual integrity and consistency between the various deliverables, at a given milestone (LCO/LCA), is critical. In particular, a system definition element should not be "incomplete" with respect to the remaining ones. For instance, if the SSRD specifies more requirements, than the architecture described in the SSAD supports, but the FRD claims that the architecture will satisfy all the requirements, the SSAD would be
considered incomplete. It is important to reconcile the deliverables, and make sure that one deliverable is not "one iteration ahead" of the other deliverables.

- The general differences between the LCO, LCA and the IOC are as follows:

**Life Cycle Objectives (LCO):**

- less structured, with information moving around
- focus on the strategy or "vision" (e.g., for the Operational Concept Description and Life Cycle Plan), as opposed to the details
- could have some mismatches (indicating unresolved issues or items)
- no need for complete forward and backward traceability
- may still include "possible" or "potential" elements (e.g., Entities, Components, …)
- some sections could be left as TBD, particularly Construction, Transition, and Support plans

**Life Cycle Architecture (LCA):**

- more formal, with solid tracing upward and downward
- no major unresolved issues or items, and closure mechanisms identified for any unresolved issues or items (e.g., "detailed data entry capabilities will be specified once the Library chooses a Forms Management package on February 15")
- no more TBDs expect possibly within Construction, Transition, and Support plans
- basic elements from the Life Cycle Plan are indicated within the Construction, Transition, and Support plans
- there should no longer be any "possible" or "potential" elements (e.g., Entities, Components, …)
- no more superfluous, unreferenced items: each element (e.g., Entities, Components, …) either should reference, or be referenced by another element. Items that are not referenced should be eliminated, or documented as irrelevant

**Initial Operating Capability (IOC):**

- complete tracings within and between models, delivered software (e.g. comments in code trace to SSAD design elements)
- MBASE models are updated to be consistent (but not necessarily complete) with delivered system, that is “as built” OCD, SSRD, SSAD, etc. models
- core system capability requirements have been implemented and tested
- at least one construction interaction
- complete set of CTS plans and reports consistent with the development completed

**For more information:** refer to the completion criteria for each deliverable, for each phase.
• The Completion Criteria for each LCO/LCA deliverable, within the LCO/LCA phase respectively, can be used as "Exit criteria". There is no mandated number of pages per se, for a deliverable. Each package should meet all the phase completion criteria, and should thus contain the pertinent information. It is generally desirable to minimize the amount of detail, through conciseness: "less is more", as long as it conveys the appropriate amount of information, and meets all the exit criteria.

• The level of detail of each section should be risk-driven. For example, interface specification between the projects should be rigorously specified, as it is very risky to leave them ambiguous. However, one should avoid premature rigorous specification of user screen layouts, as it is risky to lock these in before users have had a chance to interact with them, and GUI-builder tools make it a low risk to iterate the screens with the users.

• Use visual models (whenever possible) such as for:
  - OCD/SSRD: block diagrams, context diagrams
  - OCD/SSRD/SSAD: UML diagrams
  - LCP: tables, Gantt charts, PERT charts

• Repetition of information within the various deliverables should be discouraged, and referencing the information should be encouraged. It is not enough to make things consistent by SIMPLY repeating sections. For example, there is no need to repeat the System Requirements in the Feasibility Rationale. The feasibility rationale should establish the feasibility and consistency of the operational concept, requirements, architecture, prototypes and plans, with respect to particular (referenced) System Requirements. While redundancy, among other deficiencies, leads to lengthy and repetitious documentation and creates extra update-consistency problems, referencing items enforces traceability.

• When referencing, avoid having:
  - “broken” or invalid references (e.g., references to something, such as Project Goal, Entity, Component, etc., that does not exist)
  - “blind” or vague references (e.g., “See FRD 2.2.3”—What exactly in FRD 2.2.3 is relevant?).

• If assumptions are made in the LCO/LCA package, it is important to reality-check the assumptions as much as possible. If you say somewhere "This assumes that COTS package will do X", determine the likelihood that the assumption is true. If the likelihood is low, identify this as a risk, and determine a risk management strategy for it. Avoid introducing non-customer and non-domain-expert assumptions.

• Do not just include text from the guidelines or outside sources in your deliverables, without relating the material to your project's specifics: no need to repeat in great detail software engineering principles and explanations taken from elsewhere.

• A primary characteristic of the MBASE process is to be risk driven at all times (see MBASE invariant 5). Use this to help resolve tricky “how much should be specified” problems. Note that the assumption “more is better” and “It doesn’t hurt to have extra” often may introduce added risks (such as confusion, presumptive specification, decreased coherence and cohesion, etc.). The risk principle may often be applied as follows:
  - If it’s risky not to specify precisely, Do (e.g., a safety-critical hardware-software interface)
  - If it’s risky to specify precisely, Don’t (e.g., a GUI layout that can be easily evolved to match uncertain user needs with a GUI-builder)
B. General Formatting Guidelines

• There should be an explanation after each heading for the following subheadings: i.e., no two headings should be immediately next to each other.

• All documents should have the following information on the cover page
  • Document Title
  • Project Title
  • Team
  • Team Members and Roles
  • Date
  • Document Version Control Information

• In general, use an outline form, e.g., for Organization Activities, instead of wordy paragraphs. In an outline form, items are easier to read, and important points stand out.

• Use numbered lists as opposed to bulleted lists to be able to reference items by their number, e.g., 'Organization Goal #2', which helps traceability.

• Include captions for each figure, table, etc., to encourage referencing and enforce traceability.

C. Final Remark

We can only suggest outlines for the LCO/LCA/IOC deliverables: in particular, there is no one-size-fits-all Requirements Description, or Life Cycle Plan structure. Authors should consider all of the items in the outline. If some of them are not applicable, it should be noted as "Not applicable" or "N/A" for future reference with some justification as to why this is so. Do not feel compelled to artificially create information simply to fill out a section that is not applicable to your project. Similarly, the document outline can be expanded if there is a need. However, it is not recommended to radically change the ordering of the various sections and to freely delete critical sections. The overriding goal is clear, concise communication. Standardized guidelines help with this: if you make substantial alterations, make sure they are clear, and well justified. Haphazard documentation is a major point of project failure.

D. Conventions Used

The following conventions are used in the guidelines.

Representation:

This subsection contains descriptions of how to represent the information that is covered by the section. Each section has a text–based representation that supports tool–less development. The other representations are variants.

Trade–Offs:

This subsection contains things to think about when selecting a representation.
Guidelines for MBASE  General Guidelines

Recommendations:
This subsection contains recommendations on what or how to approach the documentation or activity.

Common Pitfalls:
This subsection contains warnings against common mistakes.

Model Integration Rules:
This subsection contains rules for maintaining consistency between the model described by the current section and the models described in other sections of the same document or other MBASE documents.

Variant Guidance
The RUP and 577 variants are usually located at the end of the applicable section. Since the main, non-variant guidelines are designed for a tool-less, text-only product, process, and property model representation, intelligent application of the variant guidelines is appropriate. Also, the variant guidelines must take precedence over the main text of the section. With these variants, keep in mind that these may not always apply well to your particular project and the extent of your use of RUP needs to be risk driven.

The order of presentation of material in a document does not necessarily imply the order of generation based on any given process or method. The MBASE documents and the order of presentation in them represent a variant (or even multiple variants). There can even be differences based on individual methods or methodologies.

RUP Guidelines:
This heading in the rest of the document indicates model variants compatible with the Rational Unified Process’ use of UML and/or using Rose UML diagrams. Since the main, non-variant guidelines are designed for a tool-less, text-only product model and the variant guidelines take precedence, do not include in text any information that is evident from a Rose diagram. This would include connectivity between actors and use-cases, or relations between classes, including multiplicity, etc.

Different Object Oriented Analysis and Design (OOA&D) approaches are often characterized with such phrases as "Use-case and Class first" or "object and behavior (operations) first". The MBASE RUP variant used in CS577 follows the Use-Case and Class first approach. However, it deviates from the Rational RUP approach to reflect the existence of legacy systems and COTS components as part of the solution.

Rational Rose calls the set of models and diagrams a model file. CS577 uses the UML constructs available in Rose on multiple diagrams to represent views of the various MBASE models, such as the Classification Model and the Enterprise Classification Model. Rose still does not support all of UML, and conventions are still appropriate, like reading and association between classes from left to right or top to bottom reading. Rose introduces the concept of a "package" which is, according to Rose's Help System, "a general-purpose model element that organizes model elements into groups. A system may be thought of as a single, high-level package, with everything else in the system contained in it." Rose also has it's own fixed, high-level packages, the "Use Case View" and the "Logical View", whose names can not be changed. Since MBASE has a much richer concept of product models for different purposes, Rose's fixed, high level packages are used as containers for MBASE-specific packages. The MBASE CS577 "Use Case View" contains the packages for the models of the OCD and the SSRD, while the "Logical View" contains the packages for the models of the SSAD. Also included are packages to facilitate the mapping of elements between and among the MBASE product models, thus allowing different levels of detail and completeness in diagrams and documents.

Rose supports the concept of Frameworks. A framework in Rational Rose is a set of predefined model elements that are needed to model a certain kind of system. An "MBASE CS577" framework is provided which is a partially populated Rational Rose model file, organized to complement the MBASE life-cycle process and models. The "MBASE CS577" framework can be added to the \Framework\Frameworks folder in your Rational Rose installation folder. The "MBASE CS577" framework contains both the package structure as well as references to RUP guidance on the diagram contents.
577 Guidelines:

This heading in the rest of the document indicates MBASE variants for CS577. Since CS577 uses Rose for UML diagramming tool for Object Oriented Analysis and Design, the 577 Guidelines include ALL the RUP GL's, even if not explicitly stated. In general, the MBASE active templates (from the EPG) also contain recommended CS577 variants that may not be indicated in the guidelines (however they are typically applicable to general classes of large software projects). Since the MBASE active templates also contain fields for the tool-less, text-only product models, they should be used with the same cautions as any software project, namely the guidelines should be tailored to the particular types and sizes of projects [in the CS577] class on a case by case, risk driven basis.

Rose Guidelines:

This heading in the rest of the document indicates Rational Rose® variants that are not specific to CS577.

UML Guidelines:

This heading in the rest of the document indicates variants based on the Object Management Group’s Unified Modeling Language’s (UML®). (Some UML variants are included under the Representation section.)
Operational Concept Description (OCD)

A. Description

1. Purpose

The purpose of the OCD is to describe how a proposed new system\textsuperscript{1} will operate within its environment. For the operational stakeholders (including users, operators, administrators, maintainers, owners, general public), it enables them to understand and refine the proposed new system. It also enables them to evolve knowledgeably from their current operational concept to the new one. For the development stakeholders, it enables them to better understand and make development decisions consistent with the operational objectives and constraints.

The OCD should describe:

- A top-level shared vision of the system’s goals and how it will achieve them;

- The reason the new system is being built, including any problems or deficiencies that the new system is intended to fix or improve;

- The concept for new system, including what it is expected to do and how it will affect the operational stakeholders;

- The success criteria and the basis for assessing the value of the new system (used to create the Business Case for the Feasibility Rationale Description (FRD));

- The structure and the dynamics of the organization\textsuperscript{2} in which the new system will be deployed, including how the people and systems will interact to accomplish the organization’s goals, and what items are used or produced by the organization;

2. Completion Criteria

The following criteria need to be satisfied at all milestones:

- The OCD must be defined using language appropriate to its primary intended audience (i.e. the operational stakeholders).

- The OCD content must be consistent with the content of the other system artifacts.

- All domain– or project–specific terms and acronyms, and any common terms with domain– or project–specific definitions must be described in “Glossary for Domain Description” (OCD 6).

The following paragraphs describe the specific completion criteria for OCD at the three project milestones.

\textsuperscript{1} Unless otherwise specified, we will simply say “the system to be developed”, “new system”, or just “the system” to describe the system that is to be developed, enhanced, updated, re–engineered, automated, or purchased.

\textsuperscript{2} We use the term organization to describe a business, government agency, association, or some subdivision thereof.
2.1 Life Cycle Objectives (LCO)

At LCO, the following items need to be described.

- Top-level system objectives and scope
  - Organization’s background and goals
  - Current structure and dynamics of the organization’s environment
  - Shortfalls of current organization’s environment
  - Capabilities of proposed system
  - System Boundary: project focus
  - System Environment
  - Evolution Considerations
- Operational concept
  - Operational stakeholders identified
  - Organizational responsibilities determined and coordinated with clients
  - Main operational scenarios coordinated with clients
  - System Concept
- Shared vision and context for stakeholders
  - Common vision and goals for system and its evolution
  - Common language and understanding of system constraints
  - Results Chain linking system development initiative with other initiatives and assumptions necessary to achieve overall system goals
  - Operational concept satisfied by at least one system/software architecture
  - Capabilities rationalized by business case analysis in Feasibility Rationale Description

2.2 Life Cycle Architecture (LCA)

At LCA, the following items need to be described.

- System objectives and scope by system increment
- Operational concept by system increment
- All critical nominal and off-nominal scenarios as coordinated with operational stakeholders
- Tracing between Project Goals, and Organization Goals and Activities
• Tracing between Capabilities and Project Goals and Organization Activities

2.3 Initial Operational Capability (IOC)

At IOC, the OCD should be consistent with the IOC versions of other MBASE artifacts, including the Transition Plan and Support Plan.

3. Intended Audience

The primary audience for the OCD consists of the operational stakeholders, including users, operators, administrators, maintainers, owners, and general public.

An important secondary audience consists of the development stakeholders, including sponsors, managers, architects, developers, testers, marketers, and developers of interfacing systems.

Recommendation:

• Use language appropriate to the primary audience.

• Add any terms that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

4. Participating Agents

The operational stakeholders should participate in definition of the operational concept, and in the associated stakeholder win-win negotiations and prototype exercises.

5. Performing Agents

The development team creates the OCD. It may contain material (mission statements, business workflows) created by the operational stakeholders.

6. High–Level Dependencies

The OCD depends on the Win–Win negotiations, which define

• Project Goals and Constraints

• System Capabilities (Priority and Rationale for proposed changes)

• System Levels of Service

• Terms for the Domain Description

The OCD provides:

• Project, Capability and Level of Service Goals for SSRD

• Domain Description and Initial Analysis for SSAD

• Stakeholder and Organizational Responsibilities for LCP

• Business Case Analysis parameters for FRD
7. Degree of Detail and Tailoring

The degree of detail in the OCD should be risk-driven (as with any MBASE model). If it’s risky to include an item (e.g., organizational relationships undergoing re-negotiation), don’t put it in. If it’s risky not to put an item (e.g., project constraints on interoperability with other systems), do put it in. Sections of the OCD may be tailored down or consolidated for small or non–critical, well defined systems.

B. Document Sections

The following subsections describe the base format for the OCD. The section headers should appear in your document. The text shown here describes the content of the section that should appear in your document.

Recommendations:

Start your document by creating a copy of the template for OCD, which can be found in the MBASE Electronic Process Guide (EPG). The OCD template includes these section headings, some draft text for some sections, and a table of contents. Replace the draft text in the template with text appropriate to your project. Tailor the template based on your project’s risks and values.

1. Introduction

1.1 Purpose of the OCD Document

Summarize the purpose and contents of this document and identify the project stakeholders. The summary shall include the following.

- The name of the system whose operational concept is described here;
- The current life cycle phase or milestone (e.g., this is the LCO version of OCD);
- A description of the system’s operational stakeholders, including the name, organization, title and role for each stakeholder;

Recommendations:

See the OCD template in the EPG for suggested wording for this section.

Common Pitfalls:

The most common mistake that new MBASE practitioners make when writing this section is that they repeat the purpose of the document from the EPG template or MBASE Guidelines instead of describing their project.

1.2 References

Provide a complete set of citations to all sources used or referenced in the preparation of this document. The citations should be in sufficient detail that the information used in this document can be traced to its source. Sources typically include books, papers, meeting notes, tools, and tool results.

577 Guidelines:

A citation should be in suitable bibliographic form. Book and papers citation should include as appropriate title of the paper, author(s), publication date, magazine or conference name, editor(s), publisher, ISBN, and an URL if available. Meeting citations should include name of meeting (if exists), date, participants, and a URL to the notes.
Tool citations should include name of the tool and its creator, version number, and a URL to descriptive material about the tool (if available). Citations to tool results should include all the information in a tool citation plus URL’s to the input data and results.

1.3 Change Summary

For each version of the OCD document, describe the main changes since the previous version. The goal is to help a reviewer focus on the most critical parts of the document needing review.

2. Shared Vision

Representation:

The few diagrams created in the following subsections can be created with either simple drawing or UML. If using UML, start a new project using your UML tool and create a UML package with the stereotype <<systemModel>> and the name “Shared Vision”. Create all models described in the following sections in this package.

Representation Note:

The guidelines in this manual for using UML put all the models and one project. Putting all the projects models in one tool project often makes it easier to represent the relations between the elements in one model and the corresponding elements in another model. For example, the relations between elements in the architecture model to their implementation. However, putting everything in one project may be difficult to manage in some tools; in which case, you may wish to create separate projects in your tool for each high–level model such as the “Shared Vision” described above.

2.1 System Capability Description

A concise description of the system that can pass the “elevator test” described in Geoffrey Moore’s Crossing the Chasm (Harper Collins, 1991, p.161). This would enable you to explain why the system should be built to an executive while riding up or down an elevator. It should take the following form:

- For (target customer)
- Who (statement of the need or opportunity)
- The (product name) is a (product category)
- That (statement of key benefit—that is, compelling reason to buy)
- Unlike (primary competitive alternative)
- Our product (statement of primary differentiation)

Here is an example for a corporate order-entry system: “Our sales people need a faster, more integrated order entry system to increase sales. Our proposed Web Order system would give us an e-commerce order entry system similar to Amazon.com’s that will fit the special needs of ordering mountain bicycles and their aftermarket components. Unlike the template-based system our main competitor bought, ours would be faster, more user friendly, and better integrated with our order fulfillment system.”
2.1.1 Benefits Realized

Many software projects fail by succumbing to the “Field of Dreams” syndrome. This refers to the American movie in which a Midwestern farmer has a dream that if he builds a baseball field on his farm, the legendary players of the past will appear and play on it (“Build the field and the players will come”).

In the *The Information Paradox* [Thorp 1998], John Thorp discusses the paradox that organizations’ success in profitability or market capitalization do not correlate with their level of investment in information technology (IT). He traces this paradox to an IT and software analogy of the “Field of Dreams” syndrome: “Build the software and the benefits will come”.

To counter this syndrome, Thorp and his company, the DMR Consulting Group, have developed a Benefits Realization Approach (BRA) for determining and coordinating the other initiatives besides software and IT system development that are needed in order for the organization to realize the potential IT system benefits. MBASE has adapted some key BRA features that help a software project and its stakeholders to develop and utilize a realistic shared vision. The most significant of these features, the Results Chain, is discussed next.

2.1.2 Results Chain

Figure 1 shows a simple results chain provided as an example in *The Information Paradox*. It establishes a framework linking Initiatives that consume resources (e.g., implement a new order entry system for sales) to Contributions (not delivered systems, but their effects on existing operations) and Outcomes, which may lead either to further contributions or to added value (e.g., increased sales). A particularly important contribution of the Results Chain is the link to Assumptions, which condition the realization of the Outcomes. Thus, in Figure 1, if order to delivery time turns out not to be an important buying criterion for the product being sold (e.g., stockable commodities such as soap or pencils), the reduced time to deliver the product will not result in increased sales.

The Results Chain provides a valuable framework by which software project members can work with clients to identify additional non-software initiatives that may be needed to realize the potential benefits enable by the software/IT system initiative. These may also identify some additional success-critical stakeholders who need to be represented and “bought into” the shared vision.
For example, the initiative to implement a new order entry system may reduce the time required to process orders only if an additional initiative to convince the sales people that the new system will be good for their careers and to train them in how to use the system effectively, is pursued. If the order entry system is so efficiency-optimized that it doesn’t keep track of sales credits, the sales people will fight using it, so increased sales may also required adding a feature to keep track of sales credits so sales people will want to use the new system.

Further, the reduced order processing cycle will reduce the time to deliver products only if additional initiatives are pursued to coordinate the order entry system with the order fulfillment system. Some classic cases where this didn’t happen were the late deliveries of Halloween candy by Hershey and the late deliveries of Christmas toys by Toys’R’Us.

Such additional initiatives need to be added to the Results Chain. Besides increasing its realism, this also identifies additional success-critical stakeholders (sales people and order fulfillment people) who need to be involved in the system definition and development process. The expanded Results Chain clarifies why these stakeholders need to be involved. It enables the organization to establish not just a stovepipe “field of dreams” software project to satisfy some requirements, but to establish a program of related software and non-software initiatives focused on value-producing end results.

577 Guidelines:

UML is not a good match for representing such processes as Results Chains. It is better to use Visio, Word, or Power Point to develop Results Chains. It is OK to stretch or approximate circles and hexagons to better accommodate text. If you are wedded to using UML and Rose, use one Static-Structure Diagram for each Results Chain. Each initiative, outcome, and assumption should be represented a classifier with a stereotype of either <<initiative>>, <<outcome>>, or <<assumption>>, as appropriate. The label of the classifier should describe the initiative, outcome, and assumption. Each assumption should be connected to one or more outcomes using a bi-directional association.

2.2 Key Stakeholders

Identify each stakeholder by their home organization, their authorized representative for project activities, and their relation to the Results Chain. The four classic stakeholders are the software/IT system’s users, customers, developers, and maintainers. Additional stakeholders may be system interfacers (the order fulfillment people above), subcontractors, suppliers, venture capitalists, independent testers, and the general public (where safety or information protection issues may be involved).

Common Pitfalls:

- Being too pushy or not pushy enough in getting your immediate clients to involve the other success-critical stakeholders. Often, this involves fairly delicate negotiations among operational organizations. If things are going slowly and you are on a tight schedule, seek the help of your higher-level managers.

- Accepting just anybody as an authorized stakeholder representative. You don’t want the stakeholder organization to give you somebody they feel they can live without. Some good criteria for effective stakeholders are that they be empowered, representative, knowledgeable, collaborative and committed.

2.3 System Boundary and Environment

The system boundary describes the services your project will be responsible for developing and delivering, and the stakeholder organizations and interfacing systems for which your project has no authority or responsibility, but with which your project must coordinate to realize a successful software/IT system and its resulting benefits.
Figure 2 shows the context diagram used to define the system boundary. It shows the type of information that may be included in a context diagram, but is not intended to be a one-size-fits-all template.

The Context Diagram for the proposed system should include entities for all the key operational stakeholders described below (OCD 2.2).

The services provided box defines the system boundary. It should just contain a list of top-level services that your system will provide. For example, besides “Order entry” in the example above, a reasonably top-level service would be an “Order authentication” service. However, lower-level decisions about details such as credit card verification or electronic signature functions should not be addressed at this stage.

Common Pitfalls:

The most common pitfalls are:

- Including system design details (design details belong in the SSAD);
- Not including all related Initiatives in the Results Chain as “Critical interfacing systems.”

577 Guidelines:

Create a Block diagram using UML’s Static-Structure Diagram that represents the system as a classifier with a stereotype <<system>> and a label that consists of the name of the system and a list of services provided by the system. Each service should have the stereotype <<service>>. Each stakeholder should be represented as an actor (e.g. a classifier with the stereotype <<stakeholder>>). If a stakeholder is a specialization of another stakeholder (e.g. “Student” and “Library User”), then show a generalization relation from the specialized stakeholder to the general stakeholder. Each stakeholder should be connected to the system with a bi-directional association. (The association is inherited by a specialized stakeholder, so an explicit association between the specialized stakeholder and the system need not be shown.)

2.4 Major Project Constraints

Summarize any constraints that are critical to the project’s success, such as:

- The project must be completed rapidly to sustain the company’s competitive edge.
- The user interface must be compatible with other company systems.
- The system must be able to adapt to changes in Internet sales tax laws.

2.4.1 Special focus: Further Shared Vision Elements for Large Systems

For small and/or rapid development projects, a top-level Results Chain, definition of stakeholders, and definition of the system’s boundary, primary services provided, and primary environmental entities are enough to get the
Inception Phase started in the right direction. For large projects, in which even the Inception phase will be a substantial commitment of stakeholders’ human and financial resources, a more substantial Inception Readiness Review (IRR) package and process is generally warranted. In the COCOMO II model [Boehm et al., 2000, p. 305], the IRR marks the beginning of the Inception phase for cost and schedule definition purposes.

For large projects, the sections 2.5 through 2.7 should be added to the Shared Vision section and reviewed during the IRR.

2.5 Top-Level Business Case

Detailed business-case guidelines are provided in Section 2.1 of Feasibility Rationale Description (FRD 2.1). For the top-level business case, it is sufficient to estimate the costs of each of the initiatives in the Results Chain, and compare them with the quantitative and qualitative benefits realized in the Results Chain outcomes.

2.6 Inception Phase Plan and Required Resources

The stakeholders committing to the Inception Phase need a reasonable estimate of how much in human and financial resources their commitment involves. They also need visibility into the major activities to be undertaken in the Inception Phase.

2.7 Initial Spiral Objectives, Constraints, Alternatives, and Risks

These will be elaborated and analyzed during the Inception Phase, but again, the stakeholders need some pre-commitment understanding of them, particularly the major risks. They should be consistent with OCD 2.4, Major Project Constraints.

3. Domain/Organization Description

If you are creating a system for a specific organization (often called “custom system”), then describe the background and goals of the organization, the organization’s current environment (including its current system), and the shortcomings of the current system. If you are creating a product for sale to a broad audience (e.g. “shrink–wrap software”), then describe the typical background, goals, environment, and shortcomings for the kind of organizations that your product is intended to serve.

This section should refine the Shared Vision described in OCD 2, and provide detailed answers to the following questions.

- Why build the system (refers to, but does not repeat results and benefits from OCD 0)?
- What are the backgrounds and goals of the Key Stakeholders (OCD 2.2)?
- What is the context in which the current system operates (may refer to OCD 2.3), including what is available to build upon, what are the processes that are currently employed, and what are the shortcomings of the current system? (Essentially “where the project is starting from?”)
- What are the critical characteristics of the current system within the organization? (Is it mission–critical? Is it a manual system? Is it a custom–built, a generic commercial–off–the–shelf (COTS) system or somewhere in between?)

The goal is to provide a common working context for all stakeholders. This context should provide sufficient detail so that the stakeholders will understand the rationale for the new system, and will be able to evaluate the goals of the new system, the effect of the new system on the organization, and the success factors for the new system.
This section should be written using terms understood by all the stakeholders in the project, especially customers and domain experts. Do not become so technical that customers and high–level managers cannot understand.

**Recommendation**

Do not describe the whole business, government agency, or association, if the new system is for a specific subdivision. Focus your description on that subdivision.

For example, in the 577 course we mostly develop systems for USC’s Information Services Division (ISD). While USC’s overall goals may include improving USC’s rank in lists of the top U.S. universities, it is too hard to relate that goal to a project that will create a web–based project management system or a multimedia archive for the ISD. The following statement would be a reasonable working statement of a goal for ISD.

“To make USC’s information services more rapidly, reliably, easily and effectively accessible to the USC community, subject to appropriate information protection, fairness, and economic constraints.”

By focusing on the appropriate organization, the relation between the organization goals and project goals will be more meaningful and straightforward. For example, since multimedia archives are usually developed for clients within the USC Libraries part of ISD, the above statement can be made more specific and relevant by substituting “library services” for “information services”.

### 3.1 Organization Background

Provide a brief overview (a few sentences) of the organization(s) sponsoring the development of this system, the organization that will be the users of the system, and the organization that will be maintaining the system. (These organizations may or may not be the same organization.)

**Recommendations**

- Consider using each group’s mission statements and/or their objectives and goals as the basis for this section.
- Do not get carried away with details that are not centrally relevant to the new system.

### 3.2 Organization Goals

Identify the broad, high–level objectives and aspirations of the sponsoring organization(s) and of the organizations that will be using and maintaining the new system. The goals should be expressed in terms of (or reference) the Benefits (OCD 2.1.1), and should only include the goals that indicate what the organization wishes to achieve by having the proposed system (e.g., increase sales, profits, and customer satisfaction).

The goals should be *measurable and relevant (M.R.)* to the organization. You should be able to answer the following questions for each goal.

- **M:** "What is a measure of this goal?"
- **R:** "Why is it relevant to the organization?"

Measures do not have to be on an absolute scale; measures relative to other measures often are more accessible. For example, “profits should be at least as high as the previous quarter”.

Goals that are never referenced by the organization processes (OCD 3.3.3 and 4.5.3), project goals (OCD 4.2) or capabilities (OCD 4.3), or system requirements (SRR 3–5) are not relevant and should be eliminated by the LCA.
Guidelines for MBASE Operational Concept Description (OCD)

Representation:

Create either:

1. A brief enumerated list of goals. To facilitate traceability, assign each goal a unique number (e.g. OG-1).
   For example:
   
   OG-1: Increase sales and profits via more efficient order processing.
   OG-2: Improve speed via faster order entry.
   OG-3: Improve quality via better in-process order visibility, reduced order defects.
   OG-4: Improve customer satisfaction via better and faster operations.

2. A set of tables, one for each goal, like Table 1.

<table>
<thead>
<tr>
<th>Goal Identifier:</th>
<th>&lt;&lt;Give a reference number&gt;&gt; such as &quot;OG-1&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization Goal:</td>
<td>&lt;&lt;Give a reference number and name&gt;&gt; such as &quot;Increase Sales and Profits&quot;</td>
</tr>
<tr>
<td>Description:</td>
<td>&lt;&lt;Describe the goal within the relevant organizations&gt;&gt; This may be deleted if the title describes it adequately, as above</td>
</tr>
<tr>
<td>Measurable:</td>
<td>&lt;&lt;Indicate how this goal is measured, perhaps within the results chain OCD 2.1&gt;&gt; such as &quot;Since sales and profits normally vary by quarter, increases will be measured with respect to the corresponding quarter in the previous year.</td>
</tr>
<tr>
<td>Relevant:</td>
<td>&lt;&lt;Describe how this goal is relevant to the organization’s success factors OCD 2.4 and background OCD 3.1&gt;&gt; such as &quot;Increased sales improve profits via increased volume and economies of scale.&quot;</td>
</tr>
</tbody>
</table>

Common Pitfalls:

- Specifying system goals, which should be documented in OCD 4.1, as organization Goals
- Not clearly specifying the measure and/or the relevance of the goals to the organization and the proposed system.
- Letting developers introduce organization goals, which should only come from interviewing customers and domain experts. The developers can help describe the M. and R.
- Specifying irrelevant goals (see discussion above).

Model Integration Rules:

- [Must be consistent with OCD 2]
- Each goal in this section should relate to one or more of the Benefits (OCD 2.1.1).
- A goal cannot violate a constraint (OCD 2.4).
- Each goal should trace to either an organization processes (OCD 3.3.3 and 4.5.3), one or more project goals (OCD 4.2) or system capabilities (OCD 4.3), and one or more system requirements (SRR 3–5).
3.3 Current Organization Environment

In the following sections, describe the current environment of the organization. (This model is sometimes called an “as is” model.) The description should include the structure, artifacts, processes, rules, and shortcomings of the organization’s environment into which the new system will be inserted. If the new system is replacing or updating a system that exists in the current environment, that system should be identified as the “current system”.

- The goals of the following subsections are:
  - To understand the structure and the dynamics of the organization in which the new system is to be deployed (the target organization).
  - To understand current problems in the target organization and identify improvement potentials.
  - To ensure that customers, end users, and developers have a common understanding of the target organization.
  - To derive the system requirements needed to support the target organization.

3.3.1 Structure

Briefly describe the current workers (e.g. people roles, systems) of the organization and the outside actors that interact with the organization (e.g. customers, suppliers, partners, prospects). Each worker and outside actor should be relevant to the current organizational goals (OCD 3.2) and the organization’s processes (OCD 3.3.3) used to achieve the goals. Identify which workers and outside actors interact with other workers and actors. Each pair that interact are said to be connected.

Representation:

Create either:

a. A Block Diagram, which is an informal notation that represents each system, person role (e.g. Customer Service Representative) and actor as a rectangle (“block”) with the name of the system, role, or actor inside. If two organizational elements (e.g. a Customer and Customer Service Representative) interact, then a line is drawn between the blocks representing the two elements. (The details of any interactions are shown in process models (OCD 3.3.3).)

b. An Organization-Architecture Model using an Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. For example, if using the Prism ADL (http://sunset.usc.edu/~softarch/Prism/), represent each system, role, and actor as a rectangle (called a “component”). If two components interact, then connect them through one or more horizontal or peer-to-peer connectors. The manual for the ADL used should be included in the References section (OCD 1.2).

c. A Business–Structure Model using the UML Static–Structure and Collaboration Diagrams. The Static–Structure Diagrams show the classes of workers and actors, and the potential relations among them. Represent each system, with the exception of the current system, and each role as a classifier with the stereotype <<business worker>>. Represent the current system, if it exists, as a classifier with the stereotype <<system>>. Represent each outside actor as a classifier with the stereotype <<business actor>>. The label of each classifier includes the name of the worker or outside actor that it represents. If

---

3 Both UML and RUP have a similar model called a “Business Object Model”, which also include artifacts (see OCD 3.3.2). However, in the UML, version 1.4, the elements are all classifiers; no objects are shown. In the RUP version, objects and classes may be shown. Since the name “Business Object Model” can cause some confusion, we use the term Business–Structure Model.
two organizational elements interact, then connect the two classifiers with a non-directional association with the stereotype <<communication>>.

If a classifier is a specialization of another classifier (e.g. Student and Library User), then show a generalization relation from the specialized classifier to the general classifier.

If there are a large number of the workers, then it is often useful to arrange the classifiers into packages that represent the sub-organization to which the classifiers belong. For example, a Customer Service Representative typically belongs to a Customer Service group within the organization. Each package should have the stereotype <<organization unit>>.

The UML Collaboration Diagrams show the particular configurations of worker and outside actor instances in the organization, possibly for a specific purpose. Each diagram shows instances of each worker and outside actor that participates in that configuration using the classifier-instance notation (i.e. classifier symbol with a label of the form “instance name : classifier name” or “: classifier name”, and with the stereotype of the named classifier). If two instances interact in this configuration, then show a link connecting the two instances.

Each Static-Structure Diagram and Collaboration Diagram should be accompanied by a brief description of its purpose.

(For simple organizations, a single Static-Structure Diagram and a single Collaboration diagram are usually sufficient.)

For each worker and outside actor identified, create a subsection numbered 3.3.1.x (see example section below) and the name of the worker or outside actor in the header. The text of the subsection should give a brief description of worker or outside actor.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

Trade-offs:

ADL’s and UML offer standardized semantics; but take longer to create then Block Diagrams. ADL’s and UML also require reviewers to know the notation or language. Some ADL’s offer semantics that support specialized analysis techniques (e.g. real-time scheduling analysis).

Recommendations:

1. Give each worker and outside actor classifier a name that expresses the responsibilities of its instances.

   • A good name is usually a noun (e.g. “librarian”), a short noun phrase (e.g. “department secretary”), or the noun form of a verb (e.g. “administrator”).

   • Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)

   • Avoid names that sound alike or are spelled alike, as well as synonyms.

   • Clear, self-explanatory names may require several words.

2. Most worker and outside actor instances will be unnamed. The instance’s label will be of the form “: classifier name”, which can be read as “some”, “a”, or “any instance of the named classifier”. For example, an instance with the label “: customer service representative” should be read as “any customer service representative”.

MBASE_Guidelines_v2.4.0.doc 19 Version Date: 2/11/2003
3. To facilitate traceability, assign each worker or outside actor classifier a unique number (e.g. Worker-1, BActor-10).

Common Pitfalls:

- Including artifacts (OCD 3.3.2) in the Business–Structure Model.
- Including workers or actors that do not participate in at least one of the organization’s processes (OCD 3.3.3), so do not help achieve an organizational goal (OCD 3.2).
- Including system components, which should be documented in the system architecture design (SSAD 3).
- Including workers or actors that are not included in the proposed structure (OCD 4.5.1), or not listed as deleted in OCD 4.5.1.3.

Model Integration Rules:

- Each worker and outside actor should be needed to achieve one or more organizational goals (OCD 3.2).
- Each operational stakeholder (OCD 3.2) that is part of the current organization should be represented as a worker; and each operational stakeholder that is not part of the current organization should be represented as an outside actor.
- Each worker and outside actor should be involved in one or more of the organization’s processes (OCD 3.3.3).
- Each worker and outside actor described here should either be included in organizational structure of the proposed system (OCD 4.5.1), or should be listed as deleted in OCD 4.5.1.3.

577 Guidelines:

Create a Business-Structure Model as described above.

3.3.1.1 Worker or Outside Actor X

Create a section at this level for each worker or outside actor classifier in the figure(s) shown in OCD 3.3.2. The header of this section should be the name of the worker or outside actor class and its unique designator, if you have assigned an identifier and it is different from the name.

Provide a brief description of its role, purpose, and responsibilities. List any attributes of the worker or outside actor that the organization’s processes (OCD 3.3.3) are known to use. List any organization processes in which the worker or outside actor participates. List all other workers or outside actors with whom this one interacts. If the worker is a system and the current instance used is a COTS product, identify the product’s name, version, and creator or distributor.

3.3.2 Artifacts

Briefly describe the current artifacts (e.g. documents, products, resources) inspected, manipulated, or produced by the organization, and the relations among the artifacts. Each artifact identified should relevant to the current organizational goals (OCD 3.2) and the organization’s processes (OCD 3.3.3) used to achieve the goals.

Your customer can give you information about the current artifacts.

- What are the major documents inspected, manipulated, or produced by the organization?
Guidelines for MBASE

- What are the resources used in production by the organization?
- What are the items produced by the organization?
- What is its general purpose, role, or description for each artifact?

**Representation**

Create either

a. A *Block Diagram*, which is an informal notation that represents each artifact as a rectangle (“block”) with the name of the artifact. If two artifacts are related, then a line should be drawn between the blocks representing the two elements. The line should be labeled with the name of the relation.

b. A *Business–Artifact Model* using the UML’s Static–Structure Diagrams, which shows the classes of artifacts and the potential relations among the artifacts. Each artifact is represented as a classifier with the stereotype <<business entity>>. If one artifact is a specialization of artifact (e.g., Tax Report and Report), then show a generalization relation from the classifier representing the specialized artifact to the classifier representing the generalized artifact.

If two artifacts are related other than by specialization, then an association is drawn to connect the two classifiers. The association should be labeled with the name of the relation. Each end of the association may be labeled with the role that artifact plays in the relation and the number of instances of the artifact that can be related to one incident of the artifact at the other end of the association.

If there are a large number of artifacts or if some artifacts are only used by certain groups within the organization, then it is often useful to arrange the artifact classifiers into packages that represent the organizational units. For example, a *Customer Record* typically belongs to a *Customer Service* group within the organization. Each package should have the stereotype <<organization unit>>.

Each Static–Structure Diagram should be accompanied by a brief description of its purpose.

For each artifact, create a subsection numbered 3.3.2.x (see example section below) and the name of the artifact in the header. The text of the subsection should give a brief description of artifact.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

**Trade–offs:**

UML provide standard semantics; but take longer to create than a Block Diagram and require reviewers to know UML.

**Recommendations:**

4. Give each classifier a name that expresses the artifact that it represents.

- A good name is usually a noun, or a short noun phrase.

---

4 Both UML and RUP have a similar model called a “Business Object Model”, which also include workers and outside actors (see OCD 3.3.1). But in the UML version the elements are all classifiers; no objects are shown. In the RUP version, objects and classes may be shown. Since the name “Business Object Model” can cause some confusion, we use the term *Business–Artifact Model*. 
• Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)

• Avoid names that sound alike or are spelled alike, as well as synonyms.

• Clear, self-explanatory names may require several words.

5. To facilitate traceability, assign artifact classifier a unique number (e.g. Artifact-01)

6. Focus on top–level artifacts, e.g. Sales Item and a Catalog of Sales Items for a Store. Additional artifacts and their details can be provided during System Analysis (SSAD 2.0).

Common Pitfalls:

• Including structural elements (OCD 3.3.1) in the Business–Artifact Model.

• Including artifacts that are not inspected, manipulated, or produced by at least one of the organizations processes (OCD 4.5.3), so are not needed to achieve an organizational goal (OCD 3.2).

• Including system components, which are structural elements that should be documented in the system architecture design (SSAD 3).

• Including artifacts that are not included in the artifacts of the proposed system (OCD 4.5.2) and are not listed as deleted in OCD 4.5.2.3.

• Including design details about the artifacts, which should be deferred to Implementation Design (SSAD 4.1)

Model Integration Rules:

• Each artifact should be needed to achieve one or more organizational goals (OCD 3.2).

• Each artifact should be used or produced by one or more of the organization’s processes (OCD 3.3.3).

• Each artifact should either be included in the artifacts of the proposed system (OCD 4.5.2), or listed as deleted in OCD 4.5.2.3.

577 Guidelines:

Create a Business–Attribute Model as described above.

3.3.2.1 Artifact X

Create a section at this level for each artifact in the figure(s) shown in OCD 3.3.2. The header of this section should be the name of the artifact class and its unique designator, if you have assigned an identifier and it is different from the name.

Provide a brief description of its role, purpose, and responsibilities. List any attributes that the organization’s processes (OCD 3.3.3) are known to use. List any organization processes that use the artifact.

3.3.3 Processes

Describe the processes that the organization used to achieve its goals (OCD 3.2). For each process, identify which workers and outside actors participate in the process, and which artifacts are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by each worker and outside actor during the process.
Guidelines for MBASE

Operational Concept Description (OCD)

A major objective of the process model is to provide a context for the business case to be developed in FRD 2.1. For example, “the proposed system will eliminate or make efficient manual order entry and verification steps”.

**Representation:**

Create either:

a. A list of process names and a story to describe each process. Each story is a paragraph describing “something the system should do” (used in Xtreme Programming [Beck 2000] and some Agile Methods).

b. A *Business Use–Case Model* is kind of *Use–Case Model* that describes the organization’s processes, the outside actors that participate in each process, and the relations among the processes and the outside actors. Represent the model as a package with the stereotype <<business use–case model>> with the package name “Current Organization Processes”. Create one or more *Use–Case Diagrams* that show the processes, the actors, and the relation among them. For each process, create a *Business Use–Case Description*, and an *Activity Diagram*.

Each organization process is represented as *use–case* with the stereotype <<business use case>>, where the name of the use–case is the name of the process. Each outside actor is represented as a classifier with the stereotype <<business actor>>, where the name of the business actor is the name of the outside actor. (Workers are not shown since they are part of the organization.) If an actor participates in a process, connect the actor and the process by a non-directional association. As in standard Use–Case Model, an actor can specialize another actor; and a process can specialize, include, or extend another process.

A *Use–Case Description* (see Figure 3) summarizes the key information about the process that the use–case represents.
Figure 3: Business Use–Case Description

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Unique identifier for traceability (e.g. Process-xx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use-Case Name</td>
<td>Name of use–case</td>
</tr>
<tr>
<td>Purpose</td>
<td>Brief description of purpose</td>
</tr>
<tr>
<td>Overview</td>
<td>Overview of the behavior</td>
</tr>
<tr>
<td>Organizational Goals:</td>
<td>List of organizational goals to which this process contributes</td>
</tr>
<tr>
<td>Priority:</td>
<td>Relative importance of the process to the business</td>
</tr>
<tr>
<td>Abstract</td>
<td>Yes or No</td>
</tr>
<tr>
<td>Actors</td>
<td>List of outside actors participating in the use–case</td>
</tr>
<tr>
<td>Pre–conditions</td>
<td>Description of state that workers, outside actors, and artifacts should be in before use-case performed. (informal text, OCL, or both)</td>
</tr>
<tr>
<td>Post–conditions</td>
<td>Description of state that workers, outside actors, and artifacts are in after use-case performed. (informal text, OCL, or both)</td>
</tr>
<tr>
<td>Specializes</td>
<td>List of use–cases that this use–case specializes</td>
</tr>
<tr>
<td>Includes</td>
<td>List of use–cases that are directly included by this use–case</td>
</tr>
<tr>
<td>Extends</td>
<td>Name of use–case extended by this use–case</td>
</tr>
<tr>
<td>Extension Points</td>
<td>List of names of extension points</td>
</tr>
</tbody>
</table>

An Activity Diagram shows what workers and outside actors participate in the process, the work they perform, and the artifacts that the process are inspects, manipulates, or produces. Each worker and outside actor is represented by a column (called a swimlane) with the name of the worker or outside actor shown at the top. Each action performed by a worker or an outside actor is shown as an activity symbol (an oval) drawn within the column representing the worker or outside actor that performs it. The label of the activity symbol describes the action.

The ordering of actions is shown by drawing transition arrows from an action to the action that follows it. Conditional behavior can be showing transitions to and from a decision state. The label of each transition from the decision state holds the description of the condition for taking that path.

Each artifact that is inspected, manipulated, or produced is shown using the classifier–instance notation (i.e. classifier symbol with a label of the form “instance name : classifier name”, “: classifier name”, “instance name : classifier name [state-name]” or “: classifier name [state-name]” and with the stereotype of the named classifier). If an action inspects an artifact, then an object flow arrow points from the artifact symbol to the activity symbol. If an action modifies or produces an artifact, then an object flow arrow points from the activity symbol to the artifact symbol.

For each process, create a subsection numbered 3.3.3.x (see example section below) with the name of the process in the header. The body of the subsection should give a description of process in the form defined by the representation chosen (e.g., stories or use–case descriptions with Activity Diagram).

---

5 UML’s formal specification language, called an Object Constraint Language (OCL).
Guidelines for MBASE Operational Concept Description (OCD)

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

Trade-offs:

A Business Use–Case Models offers standardized semantics; but takes longer to create than Stories. Business Use–Case models also require the reviewers to know the notation.

In the Business Use–Case Model, UML’s Interaction Diagrams can be used as alternative or supplement to Activity Diagrams. Interaction Diagrams show the sequence of messages exchanged among the workers and actors, while Activity Diagrams show the flow of work. Non–development stakeholders are usually familiar with the concept of workflow and not familiar with message sequencing.

Recommendations:

1. Describe only those processes that are relevant to the proposed system (e.g., processes that the proposed system will participate in).

2. Give each process a name that expresses the behavior that it represents.
   - A good name is usually a verb, or verb phrase.
   - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   - Clear, self-explanatory names may require several words.

7. To facilitate traceability, assign each process a unique number (e.g. Process-01 or BUC-01).

8. Avoid overly technical– or implementation–related processes unless they are already present in the current system. An example of an appropriate level process for an Order Entry System would be Add New Sales Item To Order.

Common Pitfalls:

- Describing a worker or customer in a process description and not describing the worker or customer in the structure of the current organization (OCD 3.3.1).
- Describing an artifact in a process description and not describing it in the artifacts of the current organization (OCD 3.3.1).
- Including processes that are not included in the processes of the proposed system (OCD 4.5.3), and are not listed as deleted in OCD 4.5.3.3.
- Including design details about the process that should be deferred to Architecture Design (SSAD 3) or Implementation Design (SSAD 4). For example, specifying exactly how user validation will be performed.

Model Integration Rules:

- Each process should contribute to the achievement of one or more organizational goals (OCD 3.2).
- Each process should contribute to one or more business cases in FRD 2.1.
• Each process should either be included in the processes model of the proposed system (OCD 4.5.3) or listed as deleted in OCD 4.5.3.3.

• Each worker or customer in a process description should be defined in the structure of the current organization (OCD 3.3.1).

• Each artifact used or produced in a process description should be defined in the artifacts of the current organization (OCD 3.3.1).

577 Guidelines:

Create a Business Use–Case Model as described above.

3.3.3.1 Process X

Create a section at this level for each process in the organization shown in the figure(s) shown in OCD 3.3.3. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Describe process’ purpose; what workers and outside actors participate in the process, and the actions they perform; and the artifacts that process inspects, manipulates, or produces.

577 Guidelines:

Create a Business Use–Case Description and one or more Activity Diagram.

3.3.4 Rules

Describe current policies (e.g. audit trails, information access, copyright protection) or constraints (inc. laws, regulations, and standards) that must be satisfied by the organization. For example, a video store may have a row that says “the customer will be fined the daily cost of the rental for videos that are returned after they are due”.

The goals are: (a) to document accurately important rules that affected the organization’s behavior, without getting into implementation details; and (b) to enable the stakeholders to raise and resolve critical issues early.

Representation:

Create a list of organization rules described in either informal text or a formal specification language (e.g. UML’s OCL).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

Trade–offs:

The informal textual representation is often less precise and more prone to error and then a formal specification. However a formal specification is may take longer to write, and requires the reviewer to know the formal specification language. (A common practice is to write an informal textual representation early in the process for reviews with non–technical stakeholders, and formal specifications for use during the analysis and reviews by technical stakeholders.)

Recommendations:

1. Describe only those rules that are relevant to the proposed system (e.g., rules that the proposed system will either have to help enforce or will constrain the implementation of the proposed system).
2. Include references to laws, regulations, standards, and Organization Policy Documents in References (OCD 1.2). Copies of which should be either include in the Appendices (OCD 7) if practical.

3. Describe each rule clearly and concisely.

4. To facilitate traceability, assign each rule a unique number (e.g. Organization-Rule-01 or OR-01).

Common Pitfalls:

- Describing a worker or customer in a rule description and not describing the worker or customer in the structure of the current organization (OCD 3.3.1).
- Describing an artifact in a rule description and not describing it in the artifacts of the current organization (OCD 3.3.1).
- Describing a process in a rule description and not describing it in the processes of the current organization (OCD 3.3.4).
- Including system rules or design rule that should described in System Analysis (SSAD 2), Architecture Design (SSAD 3), or Implementation Design (SSAD 4).

Model Integration Rules:

- Each worker or customer described in a rule should be defined in the structure of the current organization (OCD 4.5.1).
- Each artifact described in a rule should be defined in the artifacts of the current organization (OCD 3.3.1).
- Each process described in a rule should be defined in the processes of the current organization (OCD 3.3.1).

577 Guidelines:

Create a list of rules as described above using informal text.

3.3.5 Shortcomings

Describe limitations of the current organization environment and current system, if it exists. Focus on how the organization environment and current system does not fulfill the organization goals (OCD 3.2), which may have changed since the current environment and system were established, how the current system needs to be improved in supporting some of the organization processes (OCD 3.3.3), and how the current environment and system fail to address current Stakeholder Win Conditions.

Recommendations:

1. Clearly and concisely describe each shortcoming.

2. To facilitate traceability, assign each shortcoming a unique number (e.g. Shortcoming-01).

4. Proposed System

This section describes the concept and effects of the proposed system. It is the beginning of the system analysis. Specifically it addresses the following questions:
• *What* the proposed system is;

• *How well* it should perform.

Do not describe *how* it is, or will be, implemented in software (except for constraints involving mandated integrations with COTS or legacy software compatibility).

### 4.1 Statement of Purpose

Describe the purpose of the proposed system. This description should elaborate on the purpose, context, responsibilities, and organization benefits realized that are described in the Shared Vision (OCD 2). Describe how the high-level shared vision relates to the shortcomings of the current organizational environment and current system (OCD 3.7), how it fits in with the organization’s background (OCD 3.1), and goals (OCD 3.2), and how it affects the operational stakeholders (OCD 4.6.1)

**Common Pitfalls:**

• Simply listing capabilities and behaviors as statement of purpose.

• Including architectural decisions or implications (e.g., "The purpose is to design a client-server …").

• Including architectural details.

• Not relating the system purpose to the Organization Background (OCD 3.1)

**Model Integration Rules:**

• The System Capability Description (OCD 2.1) should include the key features described here.

• [Consistent with Organization Background (OCD 3.1)]

• [Consistent with Organization Goals (OCD 3.2)]

• [Consistent with Operational Stakeholders (OCD 4.7.1)]

### 4.2 Project Goals and Constraints

Describe in detail any goals and constraints that are critical to the project’s success, such as:

• The project shall be completed rapidly to sustain the company’s competitive edge.

• The user interface must be compatible with other company systems.

• The system must be able to adapt to changes in Internet sales tax laws.

• The system must be compatible with legacy code or systems.

• The software must run on the XXYY computer system.

• The software must be compatible with (or use) the ABC COTS package.

• Build the system within the budget of a gazillion US dollars. 😊
Guidelines for MBASE  Operational Concept Description (OCD)

Every project constraint described in Major Project Constraints (OCD 2.4) should be refined in this section. Additional project goals and constraints may derive from the organization goals (OCD 3.2) or the organization processes (OCD 3.3.3).

Each project goal or constraint should be measurable, relevant, and specific (M.R.S.). You should be able to answer the following questions for each goal.

M: “How is the goal measured with respect to the proposed system project?”

R: “Is this related to any Organization Goal or any external constraint?”

S: “What specific part of the system is this goal relevant to? What are the specific acceptable levels or thresholds with respect to the measures used? What specific parts of the system are to be measured?”

R: ”Why is it relevant to the organization?”

Measures do not have to be on an absolute scale; measures relative to other measures often are more accessible. Some project goals or constraints may not have a readily identifiable measure, in which case indicate how one recognizes that the project satisfied the constraint.

Many project goals apply to specific organization goals, processes, or to the infrastructure on which the system is based. Some may apply to the system as a whole. The goal description should so specify to which it applies.

Note: Project–level constraints correspond to the “constraints” in the Spiral Model cycles. Project goals are separate from system capabilities (OCD 4.3) in that goals usually affect many parts of the system, whereas capabilities relate to local and specific areas.

Representation:

Create either:

1. A brief enumerated list of goals. The description should be specific and should describe how to measure success, and how the goal relates to the organization’s goals (OCD 3.2), to its processes (OCD 3.3.3), or to the high–level view of the Major Project Constraints (OCD 2.4). To facilitate traceability, assign each goal a unique number (e.g. PG-1).

2. A set of tables, one for each goal, like Table 2.
Table 2: Project Goal M.A.R.S. Specification Form

<table>
<thead>
<tr>
<th>Project Goal:</th>
<th>&lt;&lt;Give a reference number and name&gt;&gt; such as &quot;PG-1: Limited Schedule&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>&lt;&lt;Describe this project goal&gt;&gt; e.g., “Achieve Initial Operational Capability (IOC) in 24 weeks”</td>
</tr>
<tr>
<td>Measurable:</td>
<td>&lt;&lt;Indicate how this goal can be measured with respect to the specific elements it addresses. If it is a constraint that has not easy measure, indicate what needs to be looked at within the project to see that the constraint has been adhered to &gt;&gt; E.g., “Achieving IOC means passing a Release Readiness Review”, “</td>
</tr>
<tr>
<td>Relevant:</td>
<td>&lt;&lt;Describe which organization goal (OCD 3.2), process (OCD 3.3.3), current shortcoming (OCD 3.3.5), or the major project constraint (OCD 2.4) to which this project goal is relevant &gt;&gt; e.g., “Compatible with rapid completion constraint (OCD 2.4)”</td>
</tr>
<tr>
<td>Specific:</td>
<td>&lt;&lt;Describe what particular aspects of the organization goal (OCD 3.2), process (OCD 3.3.3), current shortcoming (OCD 3.3.5), or the major project constraint (OCD 2.4) which this project goal addresses&gt;&gt; e.g., “24 weeks”. (There is no need to repeat such information if it is obvious from the above information.)</td>
</tr>
</tbody>
</table>

Common Pitfalls:

- Repeating an organization goal (OCD 3.2).
- Including a system capability (which belongs in OCD 4.2.1).
- Including a system Levels of Service goal (which belongs in OCD 4.4).
- Including a project goal that is not relevant to at least one organization goal (OCD 3.2), organization process (OCD 3.3.3), or major project constraint (OCD 2.4). (If a project goal is un-referenced but relevant, then be the appropriate earlier section.)
- Including a project goal that is not referenced by at least one project requirement (SSRD 2)

Model Integration Rules:

- Each project goal must related to one or more organization goals (OCD 3.2), processes (OCD 3.3.3), or the major project constraints (OCD 2.4).
- Each shortcoming addressed described a project goal should be described in current system shortcomings (OCD 3.3.5).
- [Must be consistent with OCD 2.1 and OCD 2.4]

4.2.1 Project Goal X

Create a section at this level for each goal of the system. The header of this section should be the name of the goal and its unique designator, if you have assigned an identifier and it is different from the name. Provide the information shown in Table 2.

Provide a brief description and its priority. List reference to organization processes that use the goal (if the goal does not appear as an activity performed by the system in the Activity Diagram for the process (OCD 4.5.3 for new or modified processes; OCD 3.3.3 for unchanged processes that are not deleted), then list the activities that the system performs which are related to this goal.
4.3 System Capabilities

Describe what the system can do, including services the operational stakeholders expect from the proposed system with respect to their organizations, including desired modifications to the current system (if it exists).

Capabilities define broad categories of system behaviors, as opposed to an operational breakdown provided by System Requirements. Capabilities should realize the high–level services described in System Boundary and Environment (OCD 2.3), and support the organization’s processes (OCD 3.3.3).

(Note: capabilities correspond with Spiral Model Objectives.)

**Representation:**

Create a *Use–Case Model* that shows the system’s capabilities, and any known relations among the capabilities. Represent the model as a package with the stereotype <<use–case model>> with the package name “System Capability”. In the package, create one or more Use–Case Diagrams. Represent each capability as *use–case* with the stereotype <<capability>>, where the name of the use–case is the name of the capability. As in standard Use–Case Diagrams, a capability can specialize, include, or extend another capability.

For each capability, create a subsection numbered 4.2.1.x (see example section below) with the name of the capability in the header. Each subsection should provide the information shown in the Table 3 for the capability.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Unique identifier (e.g. SC-xx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of capability</td>
</tr>
<tr>
<td>Description</td>
<td>What the capability allows the operational stakeholders to do</td>
</tr>
<tr>
<td>Importance</td>
<td>Relative importance of the capability (e.g. 1..n or Primary</td>
</tr>
<tr>
<td>Used In</td>
<td>Reference to organization processes that use the capability. If the capability does not appear as an activity performed by the system in the Activity Diagram for the process (OCD 4.5.3 for new or modified processes; OCD 3.3.3 for unchanged processes that are not deleted), then list the activities that the system performs which are related to this capability.</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

**Recommendations:**

1. Give each capability a name that expresses the behavior that it represents.
   - A good name is usually a verb, or verb phrase.
   - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   - Clear, self-explanatory names may require several words.

2. To facilitate traceability, assign each capability a unique number (e.g. Capability-01).
3. Capabilities should be detailed enough to be sufficiently testable that one can determine if the capability has been implemented. The following examples show the desired level of granularity or a capability.

- “Maintain up-to-date information on sales items”
- “Provide a virtual experience of touring the Doheny Library”
- “Report all leave records of the employees for a given period of time”

4. Use the “just do it” approach to eliminate the pressure to get it all right on the first pass (like writing a rough draft for a term paper). Start with the services described in OCD 2.3. Each capability may require several iterations to get right. “Go with what you know”, then revise the set of processes and adjust descriptions, as needed.

5. Describe a few capabilities then work with domain experts and operational stakeholders to clarify and refine them. As you documented capabilities, the architects will get a better idea of how the domain experts and operational stakeholders view the proposed system (i.e. the conceptual system from their perspective).

Common Pitfalls:

- Including a system requirement (which belongs in SSRD 3.2).
- Including a system Levels of Service goal (which belongs in OCD 4.4).
- Including a system behavior (which belongs in SSAD 2.3)
- Including a lot of capabilities for a small system (some of them are likely to be either system requirements or system behaviors)

Model Integration Rules:

- Each service described in OCD 2.3 should be represented as a capability. (You may have more capabilities than services; but a capability that is not listed as service must specialize a capability that is listed.)
- Each capability should be used in at least one organizational process (OCD 4.5.3). Either
  - The capability name should appear as an activity performed by the new system in the Activity Model of at least one process, or
  - The capability described here abstracts two or more lower-level operations, in which case each operation name should appear as an activity performed by the new system in the Activity Model of at least one process.

4.3.1 Capability X

Create a section at this level for each capability of the system. The header of this section should be the name of the capability and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Table 3.
4.4 Levels of Service (L.O.S.) Goals

Define the kinds of levels of service goals for the system (i.e., "how well" the system should perform a given capability) and indicate how the L.O.S. are relevant to the organization goals (OCD 3.2), capabilities (OCD 4.3) and project goals (4.2). See Appendix B for definitions for common L.O.S. attributes.

(Note: Levels of Service correspond with Spiral Model Objectives or in some cases constraints, as when the level is a non-negotiable legal requirement.

Each Levels of Service should be measurable, relevant, and specific (M.R.S.). The description of measures should specify the unit of measurement and the conditions to take the measurement under (e.g., normal operations vs. peak–load response time).

Representation:

For each L.O.S. goal, fill out the form shown in Table 4.

<table>
<thead>
<tr>
<th>Level of Service</th>
<th>&lt;&lt;Give a reference number and name&gt;&gt; such as “LS-1: Response time”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>&lt;&lt;Describe the level of service&gt;&gt;, such as “1 second desired; 2 seconds acceptable”</td>
</tr>
<tr>
<td>Measurable</td>
<td>&lt;&lt;Indicate how this goal can be measured with respect the specific elements it addresses – include as appropriate baseline measurements, minimum values, maximum values, average or typical or expected values, etc.&gt;&gt;, such as “time between hitting Enter and getting useful information on the screen”</td>
</tr>
<tr>
<td>Relevant</td>
<td>&lt;&lt;Describe which system capabilities (OCD 4.3) and perhaps project goals (4.2) this level of service is relevant to&gt;&gt;, such as “larger delays in order processing (see capability 3 in OCD 4.3) cause user frustration”</td>
</tr>
<tr>
<td>Specific</td>
<td>&lt;&lt;Describe what in particular within the system capabilities (OCD 4.3) and perhaps project goals (OCD 4.2) this level of service addresses&gt;&gt;, such as “credit card validation (in capability 3 OCD 4.3) may cause significant delay when attempting to connect to the verification service”</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

Recommendations:

1. Do not to overburden the system analysis with L.O.S. goals that the customer has not validated.

2. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

3. Do not get too hung up on measurement details. It is better to define something and refine later.

4. L.O.S. Requirements (SSRD 5) should be more specific than the L.O.S. goals specified here. If the L.O.S. goal is well defined, it is not necessary to repeat it in the SSRD. The SSRD can simply refer to the OCD definition.
Common Pitfalls:

- Including a L.O.S. goal that is of no interest to stakeholders. (Each class of stakeholders is typically concerned about specific L.O.S. goal — see Table 5).

- Including L.O.S. goals that do not reference project goals (OCD 4.2) or organization goals (OCD 3.2).

- Including project goals (which should be described in OCD 4.2).

- Including system capabilities (which should be described in OCD 4.3).

- Not satisfying the M.R.S. criteria.

- Overburdening the system with L.O.S. goals that the customer has not validated.

**Table 5: Typical Level of Service Concerns of Stakeholders**

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Roles and Primary Responsibilities</th>
<th>Level of Service Concerns</th>
<th>Primary</th>
<th>Secondary</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Public</td>
<td>Avoid adverse system side effects: safety, security, privacy</td>
<td></td>
<td>Dependability</td>
<td>Evolvability &amp; Portability</td>
</tr>
<tr>
<td>Operator</td>
<td>Avoid current and future interface problems between system and interoperating system</td>
<td></td>
<td>Interoperability, Evolvability &amp; Portability</td>
<td>Dependability, Performance</td>
</tr>
<tr>
<td>User</td>
<td>Execute cost-effective operational missions</td>
<td></td>
<td>Dependability, Interoperability, Usability, Performance, Evolvability &amp; Portability</td>
<td>Development Schedule</td>
</tr>
<tr>
<td>Maintainer</td>
<td>Avoid low utility due to obsolescence; Cost-effective product support after development</td>
<td></td>
<td>Evolvability &amp; Portability</td>
<td>Dependability</td>
</tr>
<tr>
<td>Developer</td>
<td>Avoid non-verifiable, inflexible, non-reusable product; Avoid the delay of product delivery and cost overrun.</td>
<td></td>
<td>Evolvability &amp; Portability, Development Cost &amp; Schedule, Reusability</td>
<td>Dependability, Interoperability, Usability, Performance</td>
</tr>
<tr>
<td>Customer</td>
<td>Avoid overrun budget and schedule; Avoid low utilization of the system</td>
<td></td>
<td>Development Cost &amp; Schedule, Performance, Evolvability &amp; Portability, Reusability</td>
<td>Dependability, Interoperability, Usability</td>
</tr>
</tbody>
</table>

Model Integration Rules:

- [Consistent with Organization Goals (OCD 3.2)]

- [Consistent with Level of Service Requirements (SSRD 5.)]
4.5 Changes in the Organization Environment Due to Proposed System

In the following sections describe the how the new system changes the environment of the organization. (This model is sometimes called an “to be” model) The description should include the structure, artifacts, processes, rules, and how the new system remedies the shortcomings of the organization’s current environment and system.

4.5.1 Structure

Describe how the new system will change organization’s architecture. Identify new, changed, and deleted workers and outside actors. Identify new, changed, and deleted connections. Each worker and outside actor identified should relevant to the current organizational goals (OCD 3.2), project goals (OCD 4.2), and the organization’s proposed processes (OCD 4.5.3).

Representation:

As in section OCD 3.3.1 (current organization structure), create either:

- A Block Diagram,
- An Organization-Architecture Model using Architecture Description Language, or

For each new worker or outside actor identified, create a subsection numbered 4.5.1.1.x (see example section below) and the name of the worker or outside actor in the header. The text of the subsection should give a brief description of worker or outside actor.

For each changed worker or outside actor, create a subsection numbered 4.5.1.2.x (see example section below) and the name of the worker or outside actor in the header. The text of the subsection should describe any changes.

For each deleted worker or outside actor, create a subsection numbered 4.5.1.3.x (see example section below) and the name of the worker or outside actor in the header. The text of the subsection should describe why the worker or outside actor is no longer need.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

Trade-offs:

See Trade-offs in section OCD 3.3.1 (current organization structure).

Recommendations:

1. Use same representation that you used for OCD 3.3.1 (current organization structure).

2. Highlight new or changed workers and outside actors. Some possible highlight approaches include using different colors, using different shadings, or use different font size or type.6

3. For other recommendations, see Recommendations section in Current Organization Structure (OCD 3.3.1).

---

6 Do not use italics as a highlighting technique. In UML, an italicized classifier name indicates that the class is abstract.
Common Pitfalls:

- Including artifacts (OCD 3.3.2) in this model.
- Including workers or actors that do not participate in at least one of the organization’s revised processes (OCD 4.5.3), so do not help achieve an organizational goal (OCD 3.2) and project goals (OCD 4.2).
- Including system components, which should be documented in the system architecture design (SSAD 3).

Model Integration Rules:

- Each worker and outside actor should be needed to achieve one or more organizational goals (OCD 3.2) or project goal (OCD 4.2).
- Each worker and outside actor should be involved in one or more of the organization’s processes (OCD 4.5.3).
- Each worker and outside actor described here that is not new should be described in the current organization structure (OCD 3.3.1).
- Each worker and outside actor described here as “deleted” should be described in the current organization structure (OCD 3.3.1).

Guidelines:

Create a Business-Structure Model as described in OCD 3.3.1.

4.5.1.1 New Workers and Outside actors

This section describes new workers or outside actors that appear because of the new system.

4.5.1.1.1 Worker or Outside actor X

Create a section at this level for each new worker or outside actor classifier in the figure(s) shown in OCD 3.3.2. The header of this section should be the name of the worker or outside actor class and its unique designator, if you have assigned an identifier and it is different from the name. Provide a brief description of its role, purpose, and responsibilities. List any attributes of the worker or outside actor, which are known to be used in the organization processes (OCD 4.5.3). List any organization processes in which the worker or outside actor participates. List all other workers or outside actors with whom this one interacts.

4.5.1.2 Changed Workers and Outside actors

This section describes workers or outside actors that are changed because of the new system.

4.5.1.2.1 Worker or Outside actor X

Create a section at this level for each changed worker or outside actor classifier in the figure(s) shown in OCD 3.3.2. The header of this section should be the name of the worker or outside actor class and its unique designator, if you have assigned an identifier and it is different from the name. Describe any changes to its role, purpose, responsibilities, to the attributes known to be used in the organization processes (OCD 4.5.3), to the list of organization processes in which the worker or outside actor participates, and to the list of workers and outside actors with whom this one interacts.
4.5.1.3 Deleted Workers and Outside actors

This section describes workers or outside actors that no longer participate in the organization environment because of the new system.

4.5.1.3.1 Worker or Outside actor X

Create a section at this level for each worker or outside actor classifier that appeared in the current organization structure as shown in OCD 3.3.1 and is not included in the new organization structure as shown in the figure(s) in OCD 3.3.2. The header of this section should be the name of the worker or outside actor class and its unique designator, if you have assigned an identifier and it is different from the name. Briefly described why the worker or outside actor is no longer needed.

4.5.2 Artifacts

Describe how the new system will change the organization’s artifacts. Identify new, changed, and deleted artifacts. Identify new, changed, or deleted relations. Each artifact identified should be relevant to the current organizational goals (OCD 3.2), project goals (OCD 4.2) and the organization’s proposed processes (OCD 4.5.3) used to achieve the goals.

Representation

As in section OCD 3.3.2 (current organization artifacts), create either:

a. A Block Diagram,


For each new artifact identified, create a subsection numbered 4.5.2.1.x (see example section below) and the name of the artifact in the header. The text of the subsection should give a brief description of artifact.

For each changed artifact, create a subsection numbered 4.5.2.2.x (see example section below) and the name of the artifact in the header. The text of the subsection should describe any changes.

For each deleted artifact, create a subsection numbered 4.5.2.3.x (see example section below) and the name of the artifact in the header. The text of the subsection should describe why the artifact is no longer needed.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

Trade-offs:

See Trade-offs in section OCD 3.3.2 (current organization artifacts).

Recommendations:

1. Use same representation that you used for OCD 3.3.2 (current organization artifacts).

2. Highlight new or changed artifacts. Some possible highlight approaches include using different colors, using different shadings, or us

3. For other recommendations, see Recommendations section in OCD 3.3.2 (current organization artifacts).

Common Pitfalls:

- Including workers or outside actors (OCD 3.3.2) in the Business–Structure Model.
• Including artifacts that not inspected, manipulated, or produced by at least one of the organization’s processes organization processes (OCD 4.5.3), so do not help achieve an organizational goal (OCD 3.2) and project goals (OCD 4.2).
• Including system components, which should be documented in the system architecture design (SSAD 3) and are not artifacts in any case.

Model Integration Rules:
• Each artifact should be needed to achieve one or more organizational goals (OCD 3.2) or project goals (OCD 4.2).
• Each artifact should be involved in one or more of the organization’s processes (OCD 4.5.3).
• Each artifact described here that is not new should be described in the current organization’s artifacts (OCD 3.3.1).
• Each artifact described here as “deleted” should be described in the current organization’s artifacts (OCD 3.3.1).

577 Guidelines:
Create a Business-Artifact Model as described in OCD 3.3.1.

4.5.2.1 New Artifacts
This section describes new artifacts that the new system inspects, manipulates, or produces.

4.5.2.1.1 Artifact X
Create a section at this level for each new artifact in the figure(s) shown in OCD 3.3.2. The header of this section should be the name of the artifact class and its unique designator, if you have assigned an identifier and it is different from the name. Provide a brief description of its role, purpose, and responsibilities. List any attributes that the organization’s processes (OCD 4.5.3) are known to use. List any organization processes that use the artifact.

4.5.2.2 Changed Artifacts
This section describes organization artifacts that are changed because of the new system.

4.5.2.2.1 Artifact X
Create a section at this level for each changed artifact classifier in the figure(s) shown in OCD 3.3.2. The header of this section should be the name of the artifact class and its unique designator, if you have assigned an identifier and it is different from the name. Describe any changes to its role, purpose, responsibilities, to the attributes known to be used in the organization processes (OCD 4.5.3), and to the list of organization processes that use the artifact.

4.5.2.3 Deleted Artifacts
This section describes artifacts that are no longer needed by the organization because of new system.

4.5.2.3.1 Artifact X
Create a section at this level for each artifact classifier that appeared in the current organization structure as shown in OCD 3.3.1 and is not included in the new organization structure as shown in the figure(s) in OCD 3.3.2. The header of this section should be the name of the artifact class and its unique designator, if you have assigned an identifier and it is different from the name. Briefly described why the artifact is no longer needed.
4.5.3 Processes

Describe how the new system will change the organization’s processes. Identify new, changed, and deleted processes. For each process, identify the changes to workers and outside actors that participate in the process, and to artifacts that the process inspects, manipulates, or produces; and to the actions performed by each worker and outside actor during the process.

**Representation:**

As in section OCD 3.3.3 (current organization processes), create either:

a. A list of process names & a “story” to describe each one.

b. A Business Use–Case Model

For each **new** process identified, create a subsection numbered 4.5.3.1.x (see example section below) and the name of the process in the header. Describe the process using a Use–Case Description and an Activity Diagram.

For each **changed** process, create a subsection numbered 4.5.3.2.x (see example section below) and the name of the process in the header. Describe changes to the process using a Use–Case Description and an Activity Diagram as appropriate.

For each **deleted** process, create a subsection numbered 4.5.3.3.x (see example section below) and the name of the process in the header. Describe why the process is no longer need.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

**Trade–offs:**

See Trade–offs in section OCD 3.3.3 (current organization processes).

**Recommendations:**

1. Use same representation that you used for OCD 3.3.3 (current organization processes).

2. Highlight new or changed processes. Some possible highlight approaches include using different colors, using different shadings, or us

3. For other recommendations, see Recommendations section in OCD 3.3.3 (current organization processes).

**Common Pitfalls:**

- Describing a worker or customer in a process description and not describing the worker or customer in the structure of the current organization (OCD 3.3.1).

- Describing an artifact in a process description and not describing it in the artifacts of the current organization (OCD 3.3.1).

- Including processes that do not help achieve an organizational goal (OCD 3.2) and project goals (OCD 4.2).

- Including design details about the process that should be deferred to Architecture Design (SSAD 3) or Implementation Design (SSAD 4). For example, specify exactly how user validation will be performed.
Model Integration Rules:

- Each process contribute to the achievement one or more organizational goals (OCD 3.2) or project goals (OCD 4.2).
- Each process should contribute to one or more business cases in FRD 2.1.
- Each process described here that is not new should be described in the current organization’s processes (OCD 3.3.1).
- Each process described here as “deleted” should be described in the current organization’s processes (OCD 3.3.1).
- Each worker or customer in a process description should be defined in the revised structure of the organization (OCD 4.5.1).
- Each artifact used or produced in a process description should be defined in the revised artifacts of the organization (OCD 3.3.1).

577 Guidelines:
Create a Business-Process Model as described in OCD 3.3.1.

4.5.3.1 New Processes

This section describes new processes that the new system inspects, manipulates, or produces.

4.5.3.1.1 Process X

Create a section at this level for each new process in the organization shown in the figure(s) shown in OCD 3.3.3. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name. Describe process’ purpose; what workers and outside actors participate in the process, and the actions they perform; and the artifacts that process inspects, manipulates, or produces.

577 Guidelines:
Create a Business Use–Case Description and one or more Activity Diagram.

4.5.3.2 Changed Processes

This section describes organization processes that are changed because of the new system.

4.5.3.2.1 Process X

Create a section at this level for each changed process in the organization shown in the figure(s) shown in OCD 3.3.3. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name. Describe the changes to process’ purpose; what workers and outside actors participate in the process, and the actions they perform; and the artifacts that process inspects, manipulates, or produces.

577 Guidelines:
Create a Business Use–Case Description and one or more Activity Diagram.
4.5.3.3 Deleted Processes

This section describes processes that are no longer needed by the organization because of new system.

4.5.3.3.1 Process X

Create a section at this level for each process classifier that appeared in the current organization structure as shown in OCD 3.3.1 and is not included in the new organization structure as shown in the figure(s) in OCD 3.3.2. The header of this section should be the name of the process class and its unique designator, if you have assigned an identifier and it is different from the name. Briefly described why the process is no longer needed.

4.5.4 Rules

Describe changes to the policies (e.g. audit trails, information access, copyright protection) or constraints (inc. laws, regulations, and standards) that must be satisfied by the organization. Identify new, changed, or deleted policies and constraints.

Representation:

Create lists of new, changed, and deleted organization rules described in either informal text or a formal specification language (e.g. UML’s OCL).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for Domain Description (OCD 6).

Trade–offs:

See Trade–offs in section OCD 3.3.4 (current organization rules).

Recommendations:

1. Use same representation that you used for OCD 3.3.4 (current organization rules).

2. Highlight new or changed rules. Some possible highlight approaches include using different colors, using different shadings, or us

3. For other recommendations, see Recommendations section in OCD 3.3.4 (current organization rules).

Common Pitfalls:

- Describing a worker or customer in a rule description and not describing the worker or customer in the structure of the current organization (OCD 3.3.1).
- Describing an artifact in a rule description and not describing it in the artifacts of the current organization (OCD 3.3.1).
- Describing a process in a rule description and not describing it in the processes of the current organization (OCD 3.3.4).
- Including system rules or design rule that should described in System Analysis (SSAD 2), Architecture Design (SSAD 3), or Implementation Design (SSAD 4).
Model Integration Rules:

- Each rule described here that is not new should be described in the current organization’s rules (OCD 3.3.1).
- Each rule described here as “deleted” should be described in the current organization’s rules (OCD 3.3.1).
- Each worker or customer described in a rule should be defined in the revised structure of the organization (OCD 4.5.1).
- Each artifact described in a rule should be defined in the revised artifacts of the organization (OCD 4.5.2).
- Each process described in a rule should be defined in the revised processes of the organization (OCD 4.5.3).

4.5.4.1 New Rules

List all new rules.

4.5.4.2 Changed Rules

List all new rules.

4.5.4.3 Deleted Rules

List all new rules.

4.5.5 How New System Cures Current Shortcomings

Describe how the successful development and installation of the proposed system will cure some or all the shortcomings (OCD 3.3.5) of current organization environment in accordance with the project goals (OCD 4.2), and allow the organization to meet or get closer to its goals (OCD 3.2). For each shortcoming (OCD 3.3.5) of current organization environment, describe which of the proposed system capabilities (OCD 4.3), or levels of service goals (OCD 4.4) apply.

Recommendation:

- Address each shortcoming (OCD 3.3.5).

Common Pitfalls:

- Confusing with organization goals (OCD 3.2).
- Not describing relevance to the organization background (OCD 3.1).

Model Integration Rules:

- Each shortcoming referenced here should be described in OCD 3.3.5.
- Each organization goal referenced here should be described in OCD 3.2.
- Each project goal referenced here should be described in OCD 4.2.
- For each project goal (OCD 4.2) that is relevant to a current shortcoming should be referenced here.
4.6 Effect on Organizations’ Support Operation

This section presents the effects of the proposed concept of operation and describes how the system’s operational stakeholders (users, operators, maintainers, inter-operators, managers, etc.) will interact with the system, and how they will interact with each other in the context of the system. It should elaborate upon the Results Chain defined in OCD 2.1

4.6.1 Operational Stakeholders

Describe the operational stakeholders (e.g., users, system administrator, etc.) who will interact with the new or modified system, including, as applicable, organizational structures, training/skills, responsibilities, and interactions with one another.

**Representation:**

Provide organization charts showing the responsibility relations between the various organizations involved in the software life cycle process, and identify the key responsible personnel within each organization.

- For each stakeholder, list:
  - Major activities performed by that stakeholder
  - Assumptions about user characteristics
    - Frequency of usage
    - Expected expertise (with software systems and the application domain)

**Common Pitfalls:**

- Including development–related stakeholders and organizations (e.g., designers, programmers, software maintainers, and customers.)

**Model Integration Rules:**

- [Consistent with Key Stakeholders (OCD 2.2)]
- [Consistent with proposed processes (OCD 4.5.3)]
- [Consistent with current organization processes (OCD 3.3.3)]
- [Consistent with Stakeholder Responsibilities (LCP 3.1)]

4.6.2 Organizational Relationships

Describe the relations among the management hierarchies of the system's operational stakeholders. Include the following

- Organizational Responsibilities
- Global Organization Charts
- Organizational Commitment Responsibilities
- Stakeholder Responsibilities
This serves to verify the following:

- Project scope fits within client’s authority scope or cross organizational boundaries
- Solution does not introduce organizational friction
- Solution does not shift power, confuse lines of authority, nor put outside parties on critical path for regular operational procedures

**Representation:**

Create a specialized (i.e., derived from the main organizational chart) organization chart.

**Recommendation:**

- Derived from the main organizational chart.

**Common Pitfalls:**

- Mixing class hierarchies and reporting hierarchies in an organization chart.
- Mixing people and organization units in different parts of the same organization chart (ok to put a title and a name in the same box)
- Including development–related agents and stakeholders. (The development–related stakeholders, which appear during the various phases of the project life cycle, as well as the operational stakeholders' development–related responsibilities will be defined in LCP 3.1.)

**Model Integration Rules:**

- Each operational stakeholder described here should be listed in Key Stakeholders (OCD 2.2).

### 4.6.3 Transition & Evolution

List impacts of the new operational concept on operational personnel, procedures, performance and management functions due to parallel operation of new and existing system, during transition, and likely evolution of roles and responsibilities, thereafter. Relate these to the complementary Initiatives in the Results Chain (OCD 2.1.2)

**Model Integration Rules:**

- Relate these to the complementary Initiatives in the Results Chain (OCD 2.1.2)

### 4.6.4 Operation Support

Describe anticipated organizational impacts on the user, customer, once the system is in operation. These impacts may include modification of responsibilities; addition or elimination of responsibilities or positions; need for training or retraining; and changes in number, skill levels, position identifiers, or location of personnel in various modes of operation.
5. Prototyping

This section describes the results of prototyping efforts. In particular, reference items in other areas (OCD, SSRD, LCP, etc.) that prototyping directly addresses such as requirements feasibility, COTS assessment and integration, design and schedule risks. Prototypes help with your customer negotiations:

- Reality check: are you building what the customer expected?
- A prototype gets you past “I’ll know it when I see it.”
- Makes your project concrete for the customer.
- Focuses negotiations on user concerns (when the customer isn’t the end user).

Prototypes help you design your product:

- Any gaps or inconsistencies in the design/requirements may be revealed.
- Questionable or difficult aspects can be tried out.
- Outright errors in your initial design may show up.
- Weaknesses in the development team’s skills may be revealed (in time to get training).
- Unanticipated problems with implementation technologies may be revealed (in time to try something else).
- More important or more difficult requirements or components show up; knowing about these things helps with making a reasonable schedule and division of labor.

Prototypes may be classified as:

- Non-functional (for “look and feel”):
  - Images.
  - Static interface (in some language).
  - Example interaction (series of slides, or a log or journal file).
- Functional (in various degrees):
  - Anything that runs and shows off some system features.
  - Prototypes may be classified as corresponding to phases in the development, from “Initial” to “Pre-alpha” (“Alpha” and “Beta” are industry parlance for pre-release software. An Alpha release includes major features, but isn’t intended for general use. A beta release should be mostly complete, but still needs testing.)

Prototypes may be classified by their intended use:

- A prototype might be used to demonstrate the user interface, rather than the program’s function.
- A prototype might be used to demonstrate a program’s function (in this case the UI is less important).
• Any test program written to “try out” a technology or design aspect is a prototype. Prototypes may exist only to help the development team, rather than to show to the world.

**Common Pitfalls:**

Treating prototyping as an independent modeling activity (i.e. not integrating with other MBASE models such as System Capabilities)

### 5.1 Objectives

• Describe the critical issues and risks that the prototype is attempting to resolve and the uncertainties that the prototype is trying to address

**Common Pitfalls:**

One common pitfall when prototyping is to fail to describe the prototype from the perspective of the client. In particular, the prototype should be user-oriented, and should avoid abstracting elements. It helps to use realistic sample data in the various prototype screens. E.g., use ‘Scrabble’, ‘Monopoly’, ‘Clue’, as opposed to ‘Item 1’, ‘Item 2’, ‘Item 3’.

### 5.2 Approach

Describe the type of prototypes, the stakeholders who will participate in prototyping efforts, and the development tools used. Clearly indicate how each subsection applies to each different type/instance of a prototype. For example, use a unique label for each prototype such as “[SIRSI Interface]” or use consistent subsection numbering, e.g. “5.2.1.1 [SIRSI Interface], 5.2.2.1 [SIRSI Interface]”

#### 5.2.1 Scope and Extent

Describe the type of prototypes (mock-up, functional, etc.) built and how they address the objectives stated in OCD 5.1 Explain the degree of faithfulness to the proposed system each prototype is expected to have. Describe the extent that each prototype is expected to contribute to the implementation of the proposed system.

#### 5.2.2 Participants

Describe any participation on the part of the clients in the prototyping effort: e.g., changes requested after initial evaluation

#### 5.2.3 Tools

Briefly describe:

• the tool used to develop the prototype,

• the reasons for choosing that tool,

• whether the tool met your needs,

• whether you are contemplating using a different tool.

Example:

"We started by creating a Web based prototype. But we decide to move to Microsoft Access since the system does not require public access and will be used only at the reference librarian desk".
5.2.4 Revision History

Track the prototype version, independent of the version for the overall OCD. Describe the prototype version and any changes since the last version (including changes suggested by client).

5.3 Initial Results

For each aspect of the system that you prototyped, describe the:

1. Current way of performing activity

   Example: "Currently, orders are entered via phone, email, or fax without interactive confirmation of price and availability."

2. Proposed way of performing activity

   - Include screen shot of relevant prototype screen
   - Brief explanations on how system will be used as illustrated by prototype screen (You may annotate explanations directly on screen shots)
   - You may propose multiple screens, and indicate which one your client preferred (or maybe has not decided yet which one to use).

   Example:

   Home page: Client is provided company and new-specials information, and is asked for name, account number, and indication of user type: consumer, corporate, or dealer (see screen image).

   Search Page: Client is offered the option of a single keyword search of all fields, or a more complex search (see screen image).

5.4 Conclusions

Describe how effective each prototype was in overcoming initial IKIWISI (I'll Know It When I See It) client expectations. List the following:

- The items that you will be looking into next (i.e. during the next round of prototyping) by order of priority
- The most critical risks that you hope to resolve by doing further prototyping

   Example: "Current prototype suffers from navigability problems: we will be looking into improving the usability and the navigability using frames, site maps, etc."

6. Glossary for Domain Description

Create an alphabetical listing of all uncommon or organization-specific terms, acronyms, abbreviations, and their meanings and definitions, to understand the Domain Description.

Recommendation:

- Avoid implementation technology terms at this point
• Glossary items are often answers to questions that you ask to the client: “What does this mean?”

7. Appendices

Create appendices as need. Each appendix shall be referenced in the main body of the document where the data would normally have been provided.

Include supporting documentation or pointers to electronic files containing:

• Policies (e.g., applicable Copyright Laws)
• Descriptions of capabilities of similar systems
• Additional background information
System and Software Requirements Definition (SSRD)

A. Description

1. Purpose

- Describe capability requirements (both nominal and off-nominal): i.e., the fundamental services provided by the system.

- Describe Level of Service Requirements (sometimes referred to as Non-functional requirements): i.e., the behavioral properties that the specified functions must have, such as performance, usability, etc. Level of Service Requirements should be assigned a unit of measurement.

- Describe global constraints: requirements and constraints that apply to the system as a whole. Those constraints include:
  - Interface Requirements (with users, hardware, or other systems)
  - Project Requirements (on budget, schedule, facilities, infrastructure, COTS, etc.)

- Distinguish between mandatory requirements ("must", "shall", "will"), and optional requirements ("can", "may", "could")

- All requirements must have an explicit rationale for their existence. This typically is accomplished by tracing the requirement to external model elements such as a Win–Win Agreements or Options, Prototype results, OCD System Capabilities or Goals, etc. Failure to provide an adequate rationale greatly increases risk of model clash or superfluous requirements.

- Requirements are typically tied to success models (such as Win conditions) for satisfaction and product models typically satisfy the other kinds of requirements. These are often directly related, event to the point of constraining design choices, e.g. Project Requirements: “Must use Oracle because there is existing support for it.” Design: “Use Oracle to implement Data Repository capabilities.”

Common Pitfalls:

- Specifying superfluous requirements (i.e. no critical stakeholder concurrence or rationale).

- Do not specify requirements simply because there is a sub-section for them. It is possible that some kinds of requirements do not apply to your project.

- Under specifying requirements - no clear way to implement or test

2. Completion Criteria

Below are the completion criteria for the System and Software Requirements Definition for the two phases:

- Life Cycle Objectives (Inception Phase)
2.1 Life Cycle Objectives (LCO)

- Top-level capabilities, interfaces, Level of Service levels, including:
  - Growth vectors (evolution requirements)
  - Priorities
- Stakeholders’ concurrence on essentials
- Requirements satisfiable by at least one system/software architecture

2.2 Life Cycle Architecture (LCA)

- Elaboration of capabilities, interfaces, Level of Services by iteration
  - Resolution of TBD’s (to-be-determined items)
  - Elaboration of evolution requirements
- Stakeholders’ concurrence on their priority concerns (prioritization)
- Traces to SSAD (and indirectly to FRD, LCP)
- Requirements satisfiable by the architecture in the SSAD

2.3 Initial Operational Capability (IOC)

- Update of the LCA SSRD which is compatible with the other IOC updates of the LCA package, and with the IOC Transition Plan, Support Plan, and System Qualification Testing porting of the Test Plan and Description

3. Intended audience

- Domain Expert and Customer (decision makers)
- Implementers and Architects
- Success-critical stakeholders (for their portion of the Requirements)

4. Participating Agent

Same stakeholders as Win–Win negotiation

5. Performing Agent

Cs577 team

6. High-Level Dependencies

- SSRD depends on Win–Win taxonomy
• Outline of SSRD evolves from taxonomy
• There is no one-size-fits-all taxonomy or requirements description
• Importance of adapting taxonomy to domain

• SSRD depends on OCD for:
  • Statement of Purpose
  • Project Goals and Constraints
  • Capabilities

• SSRD depends on prototype for:
  • User Interface Requirements

• SSRD depends on FRD for:
  • Changes Not Included

• Additional documents depend on SSRD:
  • SSAD to obtain (and consistency trace) System Requirements, and Project Requirements, and to support Evolution Requirements
  • LCP to relate requirement priorities to system increments or to requirements to be dropped in a design-to-cost/schedule development plan, and to check consistency with Project Requirements
  • FRD to check for satisfaction of:
    • Capability Requirements
    • System Interface Requirements
    • Level of Service Requirements
    • Evolution Requirements
  • Test Plan and Test Description and Results

7. Degree of Detail and Tailoring

• The degree of details of the SSRD should be risk-driven (as with any MBASE model). If it’s risky to put an item in (e.g., precise written specifications for GUI layouts and COTS behavior), don’t put it in. If it’s risky not to put an item in (e.g., precise written or formal specifications for safety-critical interfaces), do put it in. Sections of the SSRD may be tailored down or consolidated for small or non-critical, well defined systems. In particular all requirements must be specified with enough detail to be testable and implementable (subject to risk considerations – do not over specify or make premature or unfounded decisions). There must be some way to demonstrate that a requirement has been satisfied by the system (which will be documented in FRD). Each kind of requirement has its own testing and implementation considerations such as in indicated in the following:
  • System Capability: either supports or does not support a typical or non-trivial scenario (Use–Case)
  • Project: must have a measure, what is being measured, definition of satisfactory
B. Document Sections

1. Introduction

1.1 Purpose of the System and Software Requirements Definition Document

- Summarize the purpose and contents of this document with respect to the particular project and people involved.
- Avoid generic introductions as much as possible: for instance, you can show how your particular System and Software Requirements Definition meets the completion criteria for the given phase, and provide the necessary contributions for the systems Results Chain (OCD 2.2).

Common Pitfalls:
- Simply repeating the purpose of the document from the guidelines.

1.2 References

- Provide complete citations to prior and current related work and artifacts, documents, meetings and external tools referenced or used in the preparation of this document.
- Useful for consistency checking and traceability.

577 Guidelines:

A "complete citation" for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]

1.3 Change Summary

577 Guidelines:

For versions of the SSRD since the last ARB, include a summary of changes made in the document to ease the review process.

2. Project Requirements

- Project Requirements are general constraints and mandates placed upon the design team, as well as non-negotiable global constraints: e.g., solution constraints on the way that the problem must be solved, such as a mandated technology. Project Requirements could summarize process-related considerations from the Life Cycle Plan such as cost or Schedule constraints for a Cost/Schedule as Independent Variable process.
Project Requirements are such that, if they were left unmet, then the proposed system would not be acceptable or would not satisfy Win conditions for the success-critical stakeholders.

Project Requirements may also come from prototyping activities in which revised design considerations become mandated such as when a particular required COTS product is found to be infeasible. In such cases it may be necessary to re-negotiate Win–Win agreements or options prior to forming a requirement.

Project Requirements should be a refinement of Project Goals and Constraints (OCD 4.2): Include reference to the corresponding Project Goal or Constraint

Project Requirements should be M.A.R.S. (Measurable, Achievable, Relevant, Specific) specified in such a way that satisfaction of the requirement is testable

Defer Project Requirements about "how well" the system should perform to the Level of Service Requirements (SSRD 5)

It is not necessary to specify requirements for all the subsections listed. If there are no readily identifiable significant requirements for a particular sub-section, provide a brief rationale as to why none apply.

It is common that one requirement may fit within several sub-sections. Avoid repeating these requirements by placing each into the single category that best fits then providing references in other categories to it with perhaps adding any additional information as required to specify the requirement in that sun-section.

Example: "The system shall use the Microsoft Active Server Pages technology"

Example: "The system must have the core capabilities [specify which ones] by IOC within twelve weeks"

[Consistent with OCD 4.2]

Common Pitfalls:

Including Level of Service Requirements as Project Requirements. Those belong in SSRD 5.

Introducing Project Requirements that do not parallel or trace back from Project Goals and Constraints (OCD 4.2). One Project Goal or Constraint (OCD 4.2) may lead to several Project Requirements (SSRD 2.)

Introducing Project Requirements not negotiated with stakeholders (through Win–Win etc.)

Introducing superfluous Project Requirements that do not effect the project. In particular System Capability Requirements

Not relating each Project Requirement to the corresponding Project Goal

Not considering risk issues by either omitting critical requirements or by adding superfluous requirements

Referring only to FRD for Achievability (you may refer to FRD for rationale)

Creating superfluous or repetitious requirements simply to fill in a sub-section

Additional Guidelines:

Project Requirements should be able to answer the following Test Questions:

M: "How is the requirement measurable and testable with respect to the proposed system?"

A: "How must this requirement be achieved in the system (what are the general technology considerations)?"

R: "Is this requirement relevant to the proposed system?, "Does this requirement achieve any Project Goal?"
S: “What specifically within the relevant Project Goals and overall proposed system does this effect?”, "What are the specific details, values, or conditions that must be measured to test the satisfaction of this requirement?"

As with organization goals, to ensure Project requirement are Measurable, Achievable, Relevant, and Specific you may want to explicitly indicate these as follows:

### Table 6: M.A.R.S. Form for a Project Requirement

<table>
<thead>
<tr>
<th>Project Requirement:</th>
<th>&lt;&lt;give a reference number and name&gt;&gt;, such as “PR-1: 24 week Schedule”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>&lt;&lt;describe this project requirement&gt;&gt;, such as “The Release Readiness Review (RRR) shall be passed 24 weeks after team formation”</td>
</tr>
<tr>
<td>Measurable:</td>
<td>&lt;&lt;indicate how this requirement can be measured with respect the specific elements it addresses or project goals and constraints OCD 4.2&gt;&gt;, such as “RRR is scheduled for May 4, 2001.”</td>
</tr>
<tr>
<td>Achievable:</td>
<td>&lt;&lt;describe the top-level approach of how this requirement will be satisfied&gt;&gt;, such as “The project will use a Schedule as Independent Variable process.”</td>
</tr>
<tr>
<td>Relevant:</td>
<td>&lt;&lt;describe which project goals (OCD 4.2) this requirement is relevant to&gt;&gt;, such as “This will realize PG-15: Achieve IOC in 24 weeks.”</td>
</tr>
<tr>
<td>Specific:</td>
<td>&lt;&lt;describe what elements in particular within the project goals (OCD 4.2) this requirement addresses&gt;&gt;, such as “Implement IOC core requirements PR-1 through 10.” There is no need to repeat such information if it is absolutely obvious.</td>
</tr>
</tbody>
</table>

#### 2.1 Budget and Schedule

- Identify the available time for developing and delivering the system
- Provide the budget limits for the software and system development. Often the clients would require that existing systems be used instead of buying new ones and COTS software be used based on existing licenses. This fact should be noted in the budget requirements, as well as in the Computer Software Requirements in SSRD 2.2.5.

#### 2.2 Development Requirements

- Describe any requirements that constrain the design and implementation of the system. These requirements may be specified by reference to appropriate standards and specifications.

##### 2.2.1 Tools Requirements

- Describe any requirements that constrain the use of tools for the design and construction of the system (e.g., program generators, integrated development environments, COTS tools, etc.). Include version requirements (if applicable).

##### 2.2.2 Language Requirements

- Describe constraints on the use of particular languages for the design (e.g., UML) and the construction (e.g., Java) of the system.
2.2.3 Computer Hardware Requirements

- Describe any requirements regarding computer hardware that must be used by the system. The requirements shall include, as applicable, number of each type of equipment, type, size, capacity, and other required characteristics of processors, memory, input/output devices, auxiliary storage, communications/network equipment, and other required equipment.

2.2.4 Computer Hardware Resource Utilization Requirements

- Describe any requirements on the system's computer hardware resource utilization, such as maximum allowable use of processor capacity, memory capacity, input/output device capacity, auxiliary storage device capacity, and communications/network equipment capacity. The requirements (stated, for example, as percentages of the capacity of each computer hardware resource) shall include the conditions, if any, under which the resource utilization is to be measured.

2.2.5 Computer Software Requirements

- Describe any requirements regarding computer software that must be used by, or incorporated into, the system. Examples include operating systems, database management systems, communications/network software, utility software, input and equipment simulators, test software, and manufacturing software. The correct nomenclature, version, and documentation references of each such software item shall be provided.

2.2.6 Computer Communication Requirements

- Describe any requirements concerning the computer communications that must be used by the system. Examples include geographic locations to be linked; configuration and network topology; transmission techniques; data transfer rates; gateways; required system use times; type and volume of data to be transmitted/received; time boundaries for transmission/reception/response; peak volumes of data; and diagnostic features.

2.2.7 Standards Compliance Requirements

- Describe any particular design or construction standards that the system must comply with, and provide a reference to the standard.

Example: "The system’s object broker capabilities shall comply with the OMG CORBA standard".

2.3 Deployment Requirements

- Describe any requirements for packaging, labeling, and handling the system for delivery. These should reflect site-specific variations, and be consistent with the Transition Plan.

- Installation
  - Assumptions
  - Deployment hardware and software
  - Installer experience/skills

- Post-installation requirements
Guidelines for MBASE System and Software Requirements Definition (SSRD)

- Re-packaging
- Uninstall
- Transport and delivery

2.4 Transition Requirements

- Personnel
- Training

These should be consistent with personnel and training identified in the LCP B.3.2.3 and in the Transition Plan.

2.5 Support Environment Requirements

- Describe any required Software Support Environments to be used for the support of the delivered system
- Describe the skill levels of required support personnel and the frequency of support interactions required in terms of bug fixes, future software releases and the reporting and tracking of problems.

3. Capability Requirements

This section describes the capability requirements of the proposed system. All capability requirements must be specified in such a way that they can be implemented and tested.

3.1 System Definition

- Provide a brief overview of what the software system is. This could consist of enumerating at a high-level the various components or modules of the system.

- The System Definition should be a refinement of the Capability Description (OCD 2.1). The System Definition needs to focus on what the system does with respect to the technology that will do it, and therefore, may introduce very high-level design indications.

- [Consistent with System Capability Description (OCD 2.1)]

**Common Pitfalls:**

- Not tracing back the System Definition to the System Capability Description (OCD 2.1)
- Simply repeating the Capabilities or the System Requirements as a System Definition
- Too much detail in the System Definition

**RUP Guidelines: LCA SSRD 3.1 – System Definition**

If the SSRD is to be reviewed or used on its own, include the same system block diagram as given in the OCD §4.5, otherwise include an explicit reference.
3.2 System Requirements

- System Requirements should be a refinement of Capabilities (OCD 4.3). They need to trace from and parallel Capabilities. Each Capability must translate into at least one System Requirement (be sure to reference which one). A requirement may not directly trace back to a Capability in the OCD. In such a case the Capability Requirement may directly trace to another Capability Requirement and should be references as such (e.g. so called “supporting” capabilities which are usually design considerations). If this is not the case then the Capabilities in OCS 4.3 should be revised so long as there is sufficient rationale to justify the SSRD Capability Requirement.

- The model element integration chain (for model faithfulness) is System Capability Requirements “realize” OCD 4.3 System Capabilities that in turn “support” OCD 4.3 and 4.5.1 Domain Activities.

- System Requirements should reference related, relevant Project Goals, Levels of Service, Project Requirements, or Level of Service Requirements.

- System Requirements need to refer to high-level design specifics in the SSAD (i.e., what and how it must be implemented generally, and how the system will work).

- Requirements should describe the expected behavior when everything goes right (called “Nominal Requirements”) and how to deal with special circumstances or undesired events, errors, exceptions and abnormal conditions (called “Off-Nominal Requirements”).
  - Include Nominal Functional or Capability Requirements or Capabilities
    - During LCO, include only core/high-priority requirements
    - During LCA, add less important requirements
    - For every Capability (OCD 3.2), describe the corresponding System Requirement(s)
    - Prioritize the System Requirements, to validate that the overall life cycle strategy matches the system priorities (FRD 3.2).
    - Check that every requirement has its most critical scenarios specified in the Proposed Activities (OCD 4.5.1)
  - Include Off-Nominal Functional Requirements
    - During LCO: define high-risk off-nominal requirements; list others
    - During LCA: define moderate to high-risk off-nominal requirements; list others

Well-specified off-nominal requirements make a difference between a Byzantine system (e.g., System just fails or stops responding, or gives wrong answers, without any warning), and a fault-tolerant system (e.g., a system that gives some warning signs before failing, does an orderly shutdown, or degrades gracefully). Off-Nominal requirements may lead into additional Level of Service Requirements (Availability, Reliability...). Poorly specified off-nominal requirements are often the leading source of rework and overruns.

Example 1: "If the request cannot be completed, the server should add an entry to the error log file indicating the time the error occurred and the returned error code."

Example 2: Off-Nominal Requirements for a Business Q&A system, which allows patrons to pose queries in English, search a local database, and also runs the same query against some common search engines.
• "If the system sends a query to a remote search engine, and the remote search engine does not respond within 10 seconds, the system should timeout and try a different search engine, up to 6 different search engines."

• "If the search results exceed 1000 hits, then the system should prompt the user to refine their query instead of attempting to return all search results, which may take a very long time to process, or may overload the client machine"

Common Pitfalls:

• Requirements must be testable and specific: if one can interpret different behavioral sequences (not operational) from the statement of the requirement, the requirement is not well specified.

• Including System Requirements that do not reference the relevant Capabilities, Project Goals, Levels of Service, Project Requirements or Level of Service Requirements

• If a System Requirement traces back to multiple Capabilities, it probably indicates that you have included System Behaviors as Capabilities

• Including Level of Service Requirements ("how well the system does something") as functional System Requirements ("what the system is to do")

• Including System Requirements that do not parallel or trace back to Capabilities (OCD 4.3). One Capability may lead to several System Requirements

• Including detailed design decisions (other than development requirements) in the System Requirements. These belong in the SSAD, with the rationale for the decision in the FRD.

• Confusing between Operational Modes (see below) and sub-systems

• Confusing between Operational Modes and Off-Nominal Requirements

• Confusing between modes and states

3.2.1 Special Emphasis: Modes

• Some systems respond quite differently to similar stimulus depending on the operational mode. If that’s the case, identify the various modes, and organize the System Requirements (Nominal and Off-Nominal) around the operational modes, to avoid forgetting some critical system requirement.

• For example, a voice-operated computer system may have two operational modes:

  • Operational Mode: where the system is actually being used to perform productive work

  • Training Mode: where the operators are training the Voice Recognition module in the system to properly interpret their voice commands, to be saved for later use.

  • In operational mode, the response to the voice stimulus “Quit Application” would be to do so. In Training mode, the response might be to ask what action should be taken to the voice command ‘Quit Application’

  • A mode is a collection of state groupings within the system such that the system can only be in at most one mode at a time, there is always a well defined way to enter and exit the mode, and being in a particular mode has a significant effect on which states within a state grouping are accessible or inaccessible. E.g. an airplane in “taxi mode” can not have the state “landing gear retracted” by can have the “landing gear brakes applied” state.
The following template shows a way of organizing Section 3.2.x (this depends on whether the off-nominal requirements are also dependent on mode) around operational modes:

3.2 System Requirements
3.2.1 Mode 1
   3.2.1.1 Nominal Requirements
      3.2.1.1.1 Functional Requirement 1.1
      .
      3.2.1.1.n Functional Requirement 1.n
   3.2.1.2 Off-Nominal Requirements
      .
   3.2.2 Mode 2
   .
   .
3.2.m Mode m
   3.2.m.1 Nominal Requirements
      3.2.m.1.1 Functional Requirement m.1
      .
      3.2.m.1.n Functional Requirement m.n
   3.2.m.2 Off-Nominal Requirements
      .

4. System Interface Requirements

- In the following sections, describe any applicable requirements on how the software should interface with other software systems or users for input or output. Examples of such interfaces include library routines, token streams, shared memory, data streams, and so forth.

- Use high-level block diagrams (as applicable)

Common Pitfalls:

- Focusing only on user interface requirements and neglecting interfaces with inter-operating systems, software servers, external databases, etc…

- Providing low-level interface requirements, for systems or sub-systems which are outside of the boundary/scope of the proposed system, or which have implicit and standard interfaces (such as TCP/IP for a Web-based application)

4.1 User Interface Standards Requirements

- Describe any requirements on the various User Interfaces that the system presents to the users (who may belong to various user classes, such as end-user, programmer, etc.), which can be any of the following:
  
  a) Graphical User Interface(s) Requirements or standard style guides
  b) Command-Line Interface(s) Requirements
  c) Diagnostics Requirements

4.1.1 Graphical User Interface Standards

- Describe any Graphical User Interface (GUI) standards to which the proposed system should adhere.
• Include a few screen dumps or mockups, either directly or by reference to Operational Scenarios (OCD 4.5.3), to illustrate graphical user interface features and standards only.

• If the system is menu-driven, a description of all menus and their components should be provided.

• If the system provides tool bars, describe the tools provided

• Describe button types, shortcut commands, help features and window types in the GUI

• Identify the need for special accessibility needs of users.

Common Pitfall

• Including detailed screen shots of the prototype in the standards, these are likely to evolve and should not be included in the SSRD. They are best included in the OCD Appendix.

• Not identifying the key features required of every GUI element such as a menu, tool bars, editors etc.

4.1.2 Command-Line Interface Requirements

• Describe any Command-Line Interface (CLI) requirements

• For each command, provide:
  • Description of all arguments
  • Example values and invocations

4.1.3 Diagnostics Requirements

• Describe any requirements for obtaining debugging information or other diagnostic data

4.2 Hardware Interface Requirements

• Describe any requirements on the interfaces to hardware devices (if they are part of the system)

• Such devices include scanners, bar code readers and printers; or sensors and actuators in an embedded software system

4.3 Communications Interface Requirements

• Describe any requirements on the interfaces with any communications devices (e.g., Network interfaces) if they are part of the system

4.4 Other Software Interface Requirements

• Application Programming Interface(s) Requirements

• APIs used and provided to external systems

• Describe any requirements on the remaining software interfaces not included above

• Device drivers for special hardware devices
5. Level of Service (L.O.S.) Requirements

- Describe the desired levels of service of the System (i.e., "how well" the system should perform a given Capability Requirement)

- Level of service requirements in the SSRD should be more specific than the Levels of Service in the OCD, and indicate how they could be achieved

- Level of Service Requirements should be M.A.R.S. (Measurable, Achievable, Relevant, and Specific).

- Measures should specify the unit of measurement and the conditions in which the measurement should be taken. Where appropriate, include both desired and acceptable levels and indications on how the quality will be achieved. Note that the measure of a Level of Service need not be absolute but could be a function of another measure. E.g., if a component of the proposed system is an add-in to an existing system, an acceptable measure would be to say that the "The spell-checker add-in should not degrade the reliability of the current editor by more than 10%".

- Trace the Level of Service Requirements back to the Levels of Service and to the Organization Goals.

- To satisfy some Level of Service Requirements you may need to add to or modify the System Capabilities (and hence Capability Requirements)

- The Feasibility Rationale Description will validate (FRD 2.2.5) that the Quality Requirements are achievable with the given architecture. Do not overburden the system's design with Quality Requirements that are clearly unachievable.

- The following subsections provide possible Level of Service requirements: adapt to the project at hand, and do not feel obliged to create a requirement for each one of them.

- [Should be consistent with OCD 4.4 (Levels of Service)]

Use the following taxonomy of Level of Service requirements as a checklist. Appendix B, Level of Service Requirements, has some standard definitions for these terms.

1. Dependability
   1.1 Reliability/Accuracy
   1.2 Correctness
   1.3 Survivability/Availability
   1.4 Integrity
   1.5 Verifiability

2. Interoperability

3. Usability

4. Performance (Efficiency)

5. Adaptability
   5.1 Verifiability
   5.2 Flexibility
5.3 Expandability

5.4 Maintainability/Debuggability

6. Reusability

The M.A.R.S. criteria of a Level of Service Requirement are critical for ensuring that the requirement can be implemented and tested for satisfaction been met. To ensure Level of Service Requirements are Measurable, Achievable, Relevant, and Specific you may want to explicitly indicate these as follows:

Table 7: M.A.R.S. Form for Level of a Service Requirement

<table>
<thead>
<tr>
<th>Level of Service Requirement:</th>
<th>&lt;&lt;Give a reference number and name&gt;&gt;, such as “LR-1: 5 second search response”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>&lt;&lt;Describe this Level of Service requirement&gt;&gt;, use this to elaborate if the title does not suffice. E.g., “Maximum 5 second search response for inventory items, web site information, and order lookup”</td>
</tr>
<tr>
<td>Measurable:</td>
<td>&lt;&lt;Indicate how this requirement can be measured with respect the specific elements, include baselines, min and max tolerance, typical or expected values, etc.&gt;&gt;, such as “Response rate should be measured with respect to searching for two distinct items from the inventory, a two keyword query (AND, OR expressions only) for web site information, and a single order lookup.”</td>
</tr>
<tr>
<td>Achievable:</td>
<td>&lt;&lt;Describe briefly an approach of how this requirement is be satisfied, referring to the analysis in FRD B.2.2.5 for rationale as to why it is possible within the constraints of other requirements and the design specified in the SSAD&gt;&gt;, such as “The system will use a pre-indexed flattened attribute scheme within Sybase as described in SSAD Error! Reference source not found.. This approach avoids costly table joins and will satisfy the performance need as explained in FRD B.2.2.5.”</td>
</tr>
<tr>
<td>Relevant:</td>
<td>&lt;&lt;Describe which Capability Requirements (SSRD 3.2) this requirement is relevant to&gt;&gt;, such as “CR-4 search for inventory items, CR-07 locate information on web, CR-12 lookup order”</td>
</tr>
<tr>
<td>Specific:</td>
<td>&lt;&lt;Describe what elements in particular within the Capability Requirements (SSRD 3.2) this requirement involves&gt;&gt;, such as “Within CR-4 searching for an item by its name (a string). For CR-07 we are concerned with keyword search only (i.e. site map is excluded)”</td>
</tr>
</tbody>
</table>

Common Pitfalls:

- Simply repeating Levels of Service from OCD 3.3
- Including functional System Requirements ("what the system is to do") as Level of Service Requirements ("how well the system does something"). Note that in some areas (e.g., reliability, security, etc.), the distinction may not be very clear
- Including superfluous Level of Service Requirements not strictly negotiated with stakeholders (avoid having the developers introduce additional requirements).
- Including superfluous Level of Service Requirements that do not trace back to Levels of Service or to Organization Goals
- Including Level of Service Requirements not satisfying the M.A.R.S. criteria:

  Example of non-measurable: "The system must be as fast as possible"
Example of non-relevant: "The system should be available 24/7", for an organization that operates only 8 hours a day and does not want to perform activities beyond that. Many systems have been overloaded with requirements that are not necessary or relevant: e.g., instant response time on information used on a day-to-day basis or pinpoint accuracy when users only needed two-digit accuracy.

Example of non-specific: "The system shall be implemented as per the standards laid out by USC."

Example of non-achievable: "The system shall be available 100% of the time" for a network-based system, knowing that the network itself may not be available 100% of the time.

- Simply referring to the FRD for Achievability. Refer to the FRD for justification or rationale as to why the requirement can be achieved. The focus here is on how it is intended to be achieved.

6. Evolution Requirements

- Describe any requirements on the flexibility and expandability that must be provided to support anticipated areas of growth or changes in the proposed system or the domain itself
- Describe foreseeable directions of the system growth and change
- Describe how the software and data assets will be maintained
  - Facilities
  - Equipment
  - Service-provider relations
  - Maintenance levels
  - Maintenance cycles
  - Emergency software fixes
  - Planned software upgrade releases

Common Pitfalls:
- Evolution requirements are not simply for placing “dirt under the rug” and as with all requirements you must specify them so that they can be implemented and tested. This often implies some special consideration for the architecture to support a future requirement. For example a system may be initially required to support GIF images, but later other formats. Rather than hard-code the system to use GIF’s, a function that checks the image format then calls an appropriate display function will satisfy the evolution requirement.

6.1 Capability Evolution Requirements

- Major post-IOC capability requirements that are within the current horizon but not sufficiently important to be implemented in the initial capability.
- These requirements should be described in detail so that if the need arises, some of these can be built into the initial capability. These are often the result of limited-budget or schedule win-win negotiations
- These requirements are referred to by software and system support to plan their activities.
- [Consistent with Changes Considered (FRD B.5.1.4)]
6.2 Interface Evolution Requirements

- Describe any proposed systems with which this system must interoperate and evolve such as standards and protocols of interaction.

- How must the system adapt to interface changes?
  - Organizational changes in use on system
  - Personal changes (more, less, different style)
  - New or expanded product lines
  - Policy changes
  - Organization restructure
  - New/additional/dissolved relationships

- External systems
  - New/additional/replace system
  - Changes in external interfaces

- [Consistent with System Interface Requirements (SSRD 4)]

6.3 Technology Evolution Requirements

Describe how the desired system should address evolution of the underlying technology, how future versions of the COTS products, such as browser upgrades, new versions of server and databases, would be integrated with the proposed system and what new delivery mechanisms would be explored for the proposed system as a part of providing evolutionary support.

6.4 Environment and Workload Evolution Requirements

- Workload Characterization
  - Identify the projected growth of system usage and the scalability needs for the system.

- Data Storage Characteristics
  - As more information is stored in the system in order to support larger number of users or provide richer interactions, the storage needs of the system will grow. Identify such growth vectors.

6.5 Level of Service Evolution Requirements

Describe expectations for improvements in response time, reliability, usability, etc., as infrastructure technology improves with time. These need to be coordinated with concurrent growth in workload specified in SSRD 6.4.
7. Common Definition Language for Requirements

- Provides definitions of unfamiliar definitions, terms, and acronyms encountered or introduced during the Requirements elicitation process: the definitions express the understanding of the participants and the audience.

- No need to repeat the Common Definition Language for Domain Description (OCD 6.)

8. Appendices

- As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided.

- Include any supporting documentation
  - Detailed software and hardware specifications
  - Standards (used for compliance)
System and Software Architecture Description (SSAD)

A. Description

1. Purpose

The purpose of the SSAD is to document the results of analyzing the organizational concept of operation for the system, designing an architecture, and designing an implementation. The SSAD serves as a bridge between the Operational Concept defined during the Inception phase and the Construction phase. The SSAD is started during the Inception phase, refined during the Elaboration phase, and completed during the Construction phase.

2. Completion Criteria

The following criteria must be satisfied at all milestones.

- The OCD must be defined using language appropriate to intended audience (i.e. the operational stakeholders).
- All domain- or project-specific terms and acronyms, and any common terms with domain- or project-specific definitions must be described in “Glossary for System Analysis and Design” (SSAD 5).

The following paragraphs describe the specific completion criteria for SSAD at the three project milestones.

2.1 Life Cycle Objectives (LCO)

At LCO, the following items need to be described.

- Top-level definition of at least one feasible architecture:
  - Feasibility Criterion: a system built to the architecture would support the operational concept, satisfy the requirements, be faithful to the prototypes, and be built within the budgets and schedules in the Life Cycle Plan
  - Physical and logical elements, and relationships
    - Must provide essential features of likely components, behaviors, objects, operations
  - Choices of COTS and reusable software elements
  - Detailed analysis, high-level Design
  - Identification of infeasible architecture options

2.2 Life Cycle Architecture (LCA)

At LCA, the following items need to be described.

- Choice of architecture and iterations
• Physical and logical components, connectors, configurations, constraints
  • Must have precise descriptions of likely components, behaviors, objects, operations
• COTS and technology reuse choices
• Architectural style choices, deployment considerations
• Critical algorithms and analysis issues must be resolved
• Architecture evolution parameters
• Complete design for each component of the system
• Tracing to and from OCD/SSRD
• Assurance of satisfaction of Feasibility Criterion

2.3 Initial Operational Capability (IOC)
At IOC, the following items need to be described.
• The design of a technology-specific implementation for the system
  • Update LCA to reflect current implementation (“as built” specifications)
• Tracing to and from CTS

3. Intended audience
The audience for the SSAD consists of the following.
• Domain Expert for System Analysis
• Implementers for System Design

4. Participating Agent
The participating agents for the SSAD consists of the following.
• System Architect
• Domain Experts (to validate analysis models)
• Implementers (to validate design models)
• Project Manager (for feasibility)

5. Performing Agent
The development team creates the SSAD.
6. High-Level Dependencies

The SSAD depends on OCD for:

- Statement of Purpose
- Project Goals and Constraints
- Levels of Service
- Capabilities

The SSAD depends on SSRD for:

- System Definition
- System Requirements
- Level of Service Requirements
- System Interface Requirements
- Project Requirements

The FRD depends on SSAD to ensure satisfaction of:

- Project Requirements
- Capability Requirements
- System Interface Requirements
- Level of Service Requirements
- Evolution Requirements

7. Overall Tool Support

Use of UML and a visual modeling tool (e.g. Rational Rose) is recommended.

Avoid large, complex UML diagrams with a lot of overlapping connections. If your diagram is hard to read or overwhelmingly complex, try creating several logically related diagrams. For example, a complex Class Diagram can be simplified by one or more of the following techniques.

- Create one diagram showing classifier generalization, and one or more diagrams showing other relations
- Show only the classifier name (and possibly stereotype) in the classifier icon on diagram showing relations. Create separate diagrams, forms, or tables that show the features of each class or set of classes.

577 Guideline:

Use Rational Rose.
8. Degree of Detail and Tailoring

The degree of detail in the SSAD should be risk-driven (as with any MBASE model). If it’s risky to put an item in (e.g., detailed specifications for components, behaviors, objects, operations, etc.), don’t put it in. If it’s risky not to put in an item (e.g., precise written or formal specifications for critical or complex algorithms, configurations, COTS interfaces, etc.), do put it in. Sections of the SSAD may be tailored, eliminated, or consolidated for small or non-critical, well-defined systems.

Important note: The SSAD should not repeat information from other documents, and should reference the other information wherever applicable. Reviewing and referencing items in external documents is vital to project integration, coherence, and cohesion. Conciseness is paramount. Sloppy or non-existent references introduce a high risk of architecture mismatch with the operational concept and constraints (requirements) and implementation mismatch with the architecture and ultimately unfaithfulness of the system to the operational concept.

B. Document Sections

The following subsections describe the base format for the SSAD. The section headers should appear in your document. The text shown here describes the content of the section that should appear in your document.

Recommendations:

Start your document by creating a copy of the template for OCD, which can be found in the MBASE Electronic Process Guide (EPG). The OCD template includes these section headings, some draft text for some sections, and a table of contents. Replace the draft text in the template with text appropriate to your project. Tailor the template based on your project’s risks and values.

1. Introduction

1.1 Purpose of the SSAD Document

Summarize the purpose and contents of this document with respect to your project and the people involved. Describe how your SSAD meets the completion criteria for the current milestone.

Recommendations:

See the SSAD template in the EPG for suggested wording for this section.

Common Pitfalls:

The most common mistake that new MBASE practitioners make when writing this section is that they repeat the purpose of the document from the EPG template or MBASE Guidelines instead of describing their project.

1.2 Standards and Conventions

Describe any standards (e.g. DOD, IEEE), notations (e.g. UML), and conventions used to produce the SSAD. For example

- Standards used (DOD, IEEE)
- Notation used (UML)
• Any exceptions to the standard notation employed

• Naming Conventions

• Nouns for Components and Objects; verbs for Behaviors, Operations

• Consistent use of naming style for elements

• For example:
  
  • anObject, the_attribute, MyClass, theOperation()

1.3 References

Provide a complete set of citations to all sources used in the preparation of this document. The citations should be in sufficient detail that a reviewer can trace the information used in this document its source. Sources typically include books, papers, meeting notes, tools (referenced or used) and tool results.

577 Guidelines:

A citation should be in suitable bibliographic form. Book and papers citation should include as appropriate title of the paper, author(s), publication date, magazine or conference name, editor(s), publisher, ISBN, and an URL if available. Meeting citations should included name of meeting (if exists), date, participants, and a URL to the notes. Tool citations should include name of the tool and its creator, version number, and a URL to descriptive material about the tool (if available). Citations to tool results should include all the information in a tool citation plus URL’s to the input data and results.

1.4 Change Summary

For each version of the SSAD document, describe the main changes since the previous version. The goal is to help a reviewer know what they should emphasize.

2. System Analysis

Refine the operational concept of the proposed (OCD 4) into a model that focuses on the system and its requirements, as defined in the SRR. Describe what developers need to know in order to build the system and eliminate thing is that the developer need not worry about. The model should answer the following questions

1. Which of the organization’s workers (person roles & systems) and outside actors will this system interact with when it is operating (for software only systems, we use the term executing)?

2. What specific capabilities does this system contribute to the organization’s processes?

3. What services will this system provide, and what services will it require of other workers and of the outside actors?

4. How does the system interact with the other workers and the outside actors to implement the processes in which the system participates?

5. What are the system’s modes of operation?
6. What artifacts and information does this system inspect, manipulate, or produce? If the system uses an artifact or information when communicating with other workers or outside actors, what is the required format?

7. What rules must the system enforce or obey?

The models used in System Analysis have counterpart in the Organization/Domain Description (OCD 3) and the Proposed System (OCD 4), which provides the starting point for this analysis effort.

**Representation**

Create a UML package with the stereotype <<systemModel>> the name “System Analysis”. Create all models described in the following sections in this package.

**577 Guidelines:**

Using Rose, create a UML package with the stereotype <<systemModel>> the name “System Analysis” in the top–level package with the name Logical View.

### 2.1 Structure

Briefly describe the workers (e.g. people roles, systems) of the organization and the outside actors (e.g. customers, suppliers, partners, prospects) with which the system interacts. Identify which of the organization’s workers and outside actors interact with the system. Describe the services provided by the system and expected by the system of the workers and outside actors with which the system interacts.

Refine the structure defined in OCD 4.5.1, based on the system requirements (SSRD), by describing only the workers and outside actors with which the system interacts. Developers need not be concerned with the other workers and the outside actors, or with their interactions.

**Representation:**

Describe the context of the system using either:

a. A Block Diagram as described in OCD 3.3.1.

b. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. For example, if using the Prism ADL (http://sunset.usc.edu/~softarch/Prism/), represent each system, worker, and outside actor as a rectangle (called a “component”). If two components interact, then connect them through one or more horizontal or peer–to–peer connectors. The manual for the ADL used should be included in the References section (SSAD 1.3).

c. UML’s Static–Structure and Collaboration Diagrams. Represent the model as a package with the stereotype <<Structure Model>> and the name “System Context”. Represent the system as a classifier with the stereotype <<system>>. Represent each worker and outside actor, with which the system interacts, as a classifier with the stereotype <<actor>> (called an “actor”). The label of each classifier includes the name of the system, the worker, or the outside actor that it represents. Connect the classifier representing the system and each actor with a non–directional association that has the stereotype <<communication>>.

If an actor is a specialization of another actor (e.g. Student and Library User), then show a generalization relation from the specialized actor to the general actor. (In which case, it is unnecessary to show the association between the system and the specialized actor. The specialized actor inherits the association from the generalized actor.)

If there are a large number of the actors, then it is often useful to arrange the actors into packages that represent the sub–organization to which the actors belong. For example, a Customer Service
Representative typically belongs to a Customer Service group within the organization. Each package should have the stereotype <<organization unit>>.

The UML Collaboration Diagrams show the particular configurations of the system and actor instances, possibly for a specific purpose. Each diagram shows instances of the system and the actor that participates in that configuration using the classifier–instance notation (i.e. classifier symbol with a label of the form “instance name : classifier name” or “: classifier name”, and with the stereotype of the named classifier). If two instances interact in this configuration, then show a link connecting the two instances.

Each Static–Structure Diagram and Collaboration Diagram should be accompanied by a brief description of its purpose.

(For many systems, a single Static–Structure Diagram and a single Collaboration diagram are sufficient.)

For the system and each actor identified, create a subsection numbered 2.1.x (see example section below) and the name of the worker or outside actor in the header. The text of the subsection should give a brief description of worker or outside actor.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

**Trade–offs:**

ADL’s and UML both offer standardized semantics and require reviewers to know the notation or language. Some ADL’s offer semantics that support specialized analysis techniques (e.g. real–time scheduling analysis).

For some systems, UML’s Component Diagrams may be used instead of UML’s Static–Structure Diagram and Collaboration Diagrams. But, the current definition of UML [OMG 2001] and the support by current tools make the Component Diagram less effective for general use.

**Recommendations:**

1. Give the system classifier and each actor a name that expresses the responsibilities of its instances.

   - A good name is usually a noun (e.g. “librarian”), a short noun phrase (e.g. “department secretary”), or the noun form of a verb (e.g. “administrator”).

   - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)

   - Avoid names that sound alike or are spelled alike, as well as synonyms.

   - Clear, self-explanatory names may require several words.

2. Most system and actor instances will be unnamed. The instance’s label will be of the form “: classifier name”, which can be read as “some”, “a”, or “any instance of the named classifier”. For example, an instance with the label “: customer service representative” should be read as “any customer service representative”.

3. To facilitate traceability, assign each actor a unique number (e.g. Actor-10).

**Common Pitfalls:**

- Including artifacts (SSAD 2.2) in the context description.
Guidelines for MBASE System and Software Architecture Description (SSAD)

- Including organization workers or outside actors that do not interact with the system according to the system’s behavior description (SSAD 2.3).
- Including components of the system. (They should be documented in SSAD 3.1).
- Including workers or actors that are not included in the proposed structure (OCD 4.5.1), or are listed as deleted in OCD 4.5.1.3.

Model Integration Rules:

- Each actor should be needed to implement a system capability or interface requirement (SSRD 3 and 4).
- Each actor described here should be a worker or outside actor in organizational structure for the proposed system (OCD 4.5.1) and should be not listed as deleted in OCD 4.5.1.3.
- LCA: each actor shown here shall be used to describe the system’s behavior (SSAD 2.3).

577 Guidelines:

Use UML’s Static–Structure and Collaboration Diagrams, as described above, to define the system context.

2.1.1 System

Provide a brief description of the system’s role, purpose, and responsibilities. List the services of the system, which are used in the system’s behavior (SSAD 2.3).

Model Integration Rules:

- Each service shown in OCD 2.3 should be listed here.
- Each service should be represented as a system capability (OCD 2.3).
- The use of each service should be described in the system’s behavior (SSAD 2.3).

2.1.2 Actor X

Create a section at this level for each actor classifier in the figure(s) shown in SSAD 0. The header of this section should be the name of the worker or outside actor class and its unique designator, if you have assigned an identifier and it is different from the name.

Provide a brief description of actor’s role, purpose, and responsibilities. List any attributes of the actor that the system processes (SSAD 2.3.1) are known to be use. List any services provided by the actor that are used in a system process (SSAD 2.3.1). List any processes in which the actor participates. If the actor is a system and the current instance used is a COTS product, identify the product’s name, version, and creator or distributor.

2.2 Artifacts & Information

Briefly describe the artifacts (e.g. documents, products, resources) inspected, manipulated, or produced by the system, the kinds of information that the system must inspect, manipulate, produce, or maintain (e.g. customer record, part description, event description), and the relations among the artifacts and the information.

Refine artifact model defined in OCD 4.5.2, based on the system requirements (SSRD), by describing only the artifacts that the system inspected, manipulated, or produced by the system; by adding the main classes of information that the system must maintain; and describing the relations among the artifacts and information kinds. Developers need not be concerned with the other artifacts of the organization.
Representation

Create either

a. A Block Diagram as described in OCD 3.3.2.

b. A Business–Artifact Model as described in OCD 3.3.2. Represent the model as a package with the stereotype "<<Artifact Model>>" and the name “System Artifacts” with one or more Static–Structure Diagrams. Represent each class of information, like each artifact, as a classifier with the stereotype "<<business entity>>". (Create relations and organizational packages as in OCD 3.3.2.)

Each Static–Structure Diagram should be accompanied by a brief description of its purpose.

For each artifact and information classifier represented, create a subsection numbered 3.3.2.1x (see example section below) and the name of the artifact in the header. The text of the subsection should give a brief description of artifact or class of information.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

Trade–offs:

UML provide standard semantics; but take longer to create than a Block Diagram and require reviewers to know UML.

Recommendations:

1. Give each classifier a name that expresses the artifact or information that it represents.

   • A good name is usually a noun or short noun phrase. Avoid phrases that imply state of an object, e.g. “completed document”.
   
   • Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
   
   • Avoid names that sound alike or are spelled alike, as well as synonyms.
   
   • Clear, self-explanatory names may require several words.

2. To facilitate traceability, assign each artifact and information classifier a unique number (e.g. Artifact-01 or SA-01)

3. Focus on top–level artifacts, e.g. Sales Item and a Catalog of Sales Items for a Store. Additional artifacts and their details can be provided during architecture or implementation design (SSAD 3 or 4).

4. Focus on top–level information kinds, e.g. Customer Record and an Employee Record for a Business. Provided additional information kinds and their details (e.g. what is an Address) during architecture or implementation design (SSAD 3 or 4).

5. Information about actors (SSAD 2.1) that the system needs to maintain (e.g. user account with attributes user id and password).
Common Pitfalls:

- Including structural elements (SSAD 2.1) in the Business–Artifact Model.
- Including an artifact or information kind that the system does not inspect, manipulate, produce, or maintain according to the system’s behavior (SSAD 2.3) at LCA.
- Including an artifact or information classifiers that is not represented (called realized) in both architecture and implementation design (SSAD 3 or 4) at IOC.
- Including system components, which are structural elements that should be documented in the system architecture design (SSAD 3).
- Including artifacts that are not included in the artifacts of the proposed system (OCD 4.5.2) or listed as deleted in OCD 4.5.2.3.
- Including design details about the artifacts or information classifiers. (They should be deferred to architecture or implementation design (SSAD 3 or 4).)

Model Integration Rules:

- Each artifact and information kind should be needed to implement a system capability or interface requirement (SSRD 3 and 4).
- Each artifact should be included in the artifacts of the proposed system (OCD 4.5.2) and should not be listed as deleted in OCD 4.5.2.3.
- LCA:
  - At least one instance of each artifact and information classifier should be used to describe the system’s behavior (2.3).
  - For each system’s process (2.3) that is implemented at LCA, each artifact and information classifier used to describe that process (2.3) shall be realized in both the architecture and implementation design (SSAD 3 or 4).
- IOC:
  - Each artifact and information classifier shall be realized in both the architecture and implementation design (SSAD 3 or 4).

577 Guidelines:

Create a Business–Artifact Model as described above.

### 2.2.1 Artifact or Information Class X

Create a section at this level for each artifact in the figure(s) shown in SSAD 2.2. The header of this section should be the name of the artifact or information class and its unique designator, if you have assigned an identifier and it is different from the name.

Provide a brief description of its role, purpose, and responsibilities. List any attributes and operations that are used to describe the system’s behavior (OCD 2.3). List any system capabilities that use the artifact.
2.3 Behavior

Describe the processes that the system uses to implement its capabilities (OCD 4.3), including how it works with the actors, and how it uses the artifacts and information; and if significant, how the system’s behavior depends on its state (also called mode).

2.3.1 Processes

Describe the processes that the system uses to achieve its capabilities (OCD 4.3). For each process, identify which actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by the system and each actor during the process.

Representation:

Create a Use–Case Model that describes the system’s processes, the actors that participate in each process, and the relations among the processes and the outside actors. Represent the model as a package with the stereotype <<use–case model>> with the package name “System Processes”.

- Create one or more Use–Case Diagrams that show the capabilities, the processes that implement them, and a realization relation from each process to the capability that it implements. (Some processes may support the implementation of multiple capabilities.)

- Create one or more Use–Case Diagrams that show the processes, the actors, and the relation among them.

- For each process, create a Use–Case Description (Figure 4), and an Activity Diagram.
<table>
<thead>
<tr>
<th>Identifier</th>
<th>Unique identifier for traceability (e.g. UC-xx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use–Case Name</td>
<td>Name of use–case</td>
</tr>
<tr>
<td>Abstract</td>
<td>Yes</td>
</tr>
<tr>
<td>Purpose</td>
<td>Brief description of purpose</td>
</tr>
<tr>
<td>Actors</td>
<td>List of actors participating in the use–case</td>
</tr>
<tr>
<td>Importance</td>
<td>Relative importance of the process (e.g. 1..n or Primary</td>
</tr>
<tr>
<td>Capability:</td>
<td>List of capabilities realized</td>
</tr>
<tr>
<td>Requirements</td>
<td>List of requirements that this use–case satisfies</td>
</tr>
<tr>
<td>Risks</td>
<td>List of risks for this use–case</td>
</tr>
<tr>
<td>High–Risk?</td>
<td>Yes</td>
</tr>
<tr>
<td>Architecturally Significant?</td>
<td>Yes</td>
</tr>
<tr>
<td>Development Status</td>
<td>Draft LCO</td>
</tr>
<tr>
<td>Overview</td>
<td>Overview of the behavior</td>
</tr>
<tr>
<td>User Interface</td>
<td>Pictures and/or descriptions of user interface (e.g. reference or URL to prototype screen), if applicable, that is needed to describe the process.</td>
</tr>
<tr>
<td>Pre–conditions</td>
<td>Description of the state that system and each participant should be in before use–case performed. (informal text, OCL, or both)</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>Description of the system's and participants' states after use–case performed. (informal text, OCL, or both)</td>
</tr>
<tr>
<td>Specializes</td>
<td>List of use–cases that this use–case specializes</td>
</tr>
<tr>
<td>Includes</td>
<td>List of use–cases that are directly included by this use–case</td>
</tr>
<tr>
<td>Extends</td>
<td>Name of use–case extended by this use–case</td>
</tr>
<tr>
<td>Extension Points</td>
<td>List of names of extension points</td>
</tr>
</tbody>
</table>
Typical Course of Action

<table>
<thead>
<tr>
<th>Seq. #</th>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Description of actor action</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Description of system response</td>
</tr>
<tr>
<td>3</td>
<td>Description of actor action</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Description user response</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alternate Course of Action: Name

<table>
<thead>
<tr>
<th>Seq. #</th>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Description of actor action</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Description of system response</td>
</tr>
<tr>
<td>3</td>
<td>Description of actor action</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Description user response</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exceptional Course of Action: Name

<table>
<thead>
<tr>
<th>Seq. #</th>
<th>Actor Actions</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Description of actor action</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Description of system response</td>
</tr>
<tr>
<td>3</td>
<td>Description of actor action</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Description user response</td>
</tr>
<tr>
<td>etc.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each process, create a subsection numbered 2.3.1.x (see example section below) with the name of the process in the header. Each subsection should provide the information shown in the Figure 4 and an Activity Diagram.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. Create a use-case for each process needed to implement a capability. For example, the capability Maintain Vendor Profile may require processes: Add Vendor, Delete Vendor, and Change Vendor.

2. Use the “just do it” approach to eliminate the pressure to get go right set of processes on the first pass (like writing a rough draft for a term paper). Start with the system’s actions used in the organization’s processes that are associated with each capability (see the “Used In” field of the capabilities description). Each process may require several iterations to get right. “Go with what you know”, then revise the set of processes and adjust descriptions, as needed.
3. Give each process a name that expresses the behavior that it represents.

- A good name is usually a verb or verb phrase.
- Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
- Avoid names that sound alike or are spelled alike, as well as synonyms.
- Clear, self-explanatory names may require several words.

4. To facilitate traceability, assign each process a unique number (e.g. Process-01 or UC-01).

5. Each process should be described in enough detail to be testable.

6. For LCO, the following fields of Use–Case Description should be filled out for all use–cases: Name, Purpose, Actors, Importance, Requirements, Development Status, Pre–conditions, Post–conditions, and Includes.

7. For LCA, Typical, Alternate, and Exceptional Courses of Actions should be described for high–risk, architecturally significant, or particularly complex use–case. Fill–in the Includes & Extension Points fields if the courses of actions specify the inclusion of other use–cases or extension point, respectively.

8. For IOC, use–cases descriptions should be completely filled out for all use–cases.

Common Pitfalls:

- Simply creating one process for each capability (many capabilities will require more than one process since the capabilities were described a high-level, e.g. Maintain Vendor Profile).
- Confusing an Alternate or Exceptional Course of Action with a separate process.
- Describing a pre–condition then testing for that condition early in the Course of Action.
- Describing a post–condition than testing for that condition at the end of the Course of Action.
- Describing a condition has a precondition or post–condition when the condition must be tested in the Course of Action.

Model Integration Rules:

- Each system service described in SSAD 2.1.1 must be represented as a capability. (You may have more capabilities than services; but a capability that is not listed as service must specialize capability that is listed.)
- Each system capability described in OCD 4.3 must be represented as a capability here and must have at least one process that realizes it.
- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1.1).
- Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information of the system (SSAD 2.2).
2.3.1.1 Process X

Create a section at this level for each process of the system. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.

2.3.2 Modes of Operation

Describe the state behavior (also called modal behavior) of the system, i.e. how the behavior of the system depends on the state (sometimes called mode) it is in. Describe the high-level states of the system, the events that cause the system to change modes, and how the processing is different in each mode.

Some systems respond differently to the same stimulus depending on the operational mode. For example, a voice-operated computer system may have two operational modes:

- **Operational Mode**: where the operators use the system to perform productive work.
- **Training Mode**: where the operators train the system to interpret their pronunciation of the commands to be used in the Operational Mode.

In **Operational Mode**, the voice-operated computer system would respond to the voice stimulus “Quit Application” would be to do so. In **Training Mode**, the system response to the same stimulus might be to ask the operator what action should be taken when “Quit Application” is spoken while in **Operational Mode**.

The term **mode** is often used for the high-level states of a system, which control what the system does in a broad way. (Low-level states typically control small details of the operation.) The mode defines a set of states that the system can be in and another that it cannot. For example, an airplane in taxi state (“mode”) can not be in the state landing gear retracted; but can be in the landing gear brakes applied and airbrakes applied states.

For simple systems, the system can only be in at most one state (mode) at a time. More complex systems may have nested and concurrent sub-states (sub-modes). For example, an airplane in the taxi state (“mode”) may be in the landing gear brakes applied and airbrakes applied substates simultaneously.

**Representation:**

Create a **State Model** that describes the high-level modes of the system, the events that cause the system to change modes, and how the processing is different in each mode. Represent the model as a package with the stereotype <<state model>> with the package name “System Modes”. (Some UML tools allow you to attach the State Model to the classifier representing a system.). Create one or more UML Statecharts that show the modes of system and the events that cause mode changes.

Represent each mode as a state. If an event causes a mode change, then draw a transition line from the mode system is in when the event occurs to the mode to system will enter. Label the transition with the name of the event that must occur and any condition that must be met for the mode change to occur. Indicate the state in which the system begins (initial state) by placing the initial pseudostate on the diagram, and by drawing a transition from the initial pseudostate to initial state. If there is a final state, show it as an unlabeled “bull’s eye” symbol, and add appropriate transitions from other states to the final state.

For each mode, create a subsection numbered 2.3.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list system capabilities (OCD 4.3) and processes (SSAD 2.3.1) are available in that mode (see Table 8). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode-specific behavior).
Table 8: Available Capabilities & Processes in Mode

<table>
<thead>
<tr>
<th>Capability id &amp; name.</th>
<th>Processes</th>
<th>Mode Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process id &amp; name</td>
<td>Describe any mode-specific behavior or L.O.S. goals</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right-click on the system classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Model” that is attached to the system classifier, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

Recommendations:

1. Each state should have clearly defined entry and exit events. State models should be deterministic, i.e. if you are in state A and an event occurs, then it should be clear what state will become the current state. State models can be non-deterministic; but it is not recommended.

2. Keep the state model simple (i.e. only high-level modes) unless your system has high safety or reliability requirements.

3. Give each state a name that expresses the situation that it represents.
   
   - A good name is the present participle form of a verb (e.g. “waiting”, “taxing”, “flying”) or the past participle form of a verb (e.g. “stopped”, “completed”), or an adjective (e.g. “active”, “passive”, “draft”, “final”).
   
   - Each name must be unique relative to the containing state model. (Two states in different models or substates in different states.)
   
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   
   - Clear, self-explanatory names may require several words.

4. Give each event a name that expresses the “thing that happened”. Events include asynchronous stimuli between two instances (signals), operation invocations (calls), the passage of time, or a change in state. Events may be synchronous or asynchronous.
   
   - A good name for a signal or for call is the name of the operation being requested.
   
   - Each name must be unique relative to the containing system model.
   
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   
   - Clear, self-explanatory names may require several words.
Guidelines for MBASE System and Software Architecture Description (SSAD)

- A time event should be of the form “after (x units)”, “when (date = xyyyy))”, or “when (time = xyyyy)”. 
- A change of state should be of the form “when (boolean expression)”.

Common Pitfalls:

- Not specifying the initial state for a model.
- Confusing a state with an activity.
- Confusing an event with a condition.

Model Integration Rules:

- Each required mode (SSRD 3.2.1) should be represented as a mode here.
- Each requirement associated with are required mode (SSRD 3.2.1) should be realized by one or more of the processes associated with that mode.
- Each system capability (OCD 4.3) should be listed as available in one or more modes.
- Each system process (SSAD 2.3.1) should be listed as available in one or more modes.
- Each capability listed as available in one or more modes should appear in described in OCD 4.3.
- Each process listed as available in one or more modes should appear in described in SSAD 2.3.1.

2.3.2.1 Mode X

Create a section at this level for each mode of the system. The header of this section should be the name of the mode and its unique designator, if you have assigned an identifier and it is different from the name. Describe the mode (e.g. for the airplane's taxi mode, “when in this mode, the airplane is moving on the ground, usually to or from a gate”).

List in a table the system capabilities (OCD 4.3), the processes (SSAD 2.3.1) associated with each capability that are available in this mode; describe any modal effects on the processes. For example,

<table>
<thead>
<tr>
<th>Capability</th>
<th>Processes</th>
<th>Modal Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability-01</td>
<td>Process-01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Process-02</td>
<td>LOS-01 only applies to this process in this mode.</td>
</tr>
<tr>
<td>Capability-03</td>
<td>Process-05</td>
<td></td>
</tr>
</tbody>
</table>

2.4 L.O.S. Goals

Describe how the system L.O.S. requirements (SSRD 5) apply to specific system’s processes and modes, or to information classifiers that the system. Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).
Representation:

Create a table like Table 9, that lists each L.O.S. requirements (SSRD 5), what processes, modes, or information classifiers to which the L.O.S. requirement applies, and if the L.O.S. applies to more than one element, describe how the L.O.S. requirement applies. Table 10 shows are representative sample map.

Table 9: Map of L.O.S. Requirements to Goals for System Processes, Modes, and Information

<table>
<thead>
<tr>
<th>L.O.S. Requirement</th>
<th>Applies To</th>
<th>How</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name &amp; Identifier</td>
<td>List of processes, modes, or artifact or information classifiers</td>
<td>If L.O.S. requirement applies to more than one element, describe how it applies.</td>
</tr>
</tbody>
</table>

Table 10: Sample Map of L.O.S. Requirements to Goals for System Processes, Modes, and Information

<table>
<thead>
<tr>
<th>L.O.S. Requirement</th>
<th>Applies to:</th>
<th>How</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoS-01 ...</td>
<td>Process-42 ... Process-55 ...</td>
<td>LoS-01 is average of values for each process</td>
</tr>
<tr>
<td>LoS-02 ...</td>
<td>All processes</td>
<td>Equally</td>
</tr>
<tr>
<td>LoS-03 ...</td>
<td>Mode-05 ...</td>
<td></td>
</tr>
<tr>
<td>LoS-04 ...</td>
<td>SA-06 Customer Record SA-07 Supplier Record</td>
<td>75% to customers, 25% to suppliers</td>
</tr>
</tbody>
</table>

As shown in Table 10, when an L.O.S. requirement applies to multiple elements, the L.O.S. requirement generally applies to the elements in one of three ways.

- The requirement applies the same way to the elements (i.e. “equally”, “uniformly”).
- The requirement applies in equal portions to the elements. The requirement value is the average of the value for each element.
- The requirement applies in unequal portions to the elements. The requirement value is typically the weighted average of the value for each element.

For each L.O.S. requirement that is the average or weighted average of the values for more than system element, create a subsection numbered 2.3.1.x (see example section below) with the name of the L.O.S. requirement in the header. Fill in a L.O.S. Form (Table 11) to describe the goal for each system element to which the L.O.S. requirement applies.
**Table 11: Level of Service Specification Form**

<table>
<thead>
<tr>
<th>Level of Service</th>
<th>&lt;&lt;Give a reference number and name&gt;&gt; such as “LS-1.1: Response time”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>&lt;&lt;Describe the level of service&gt;&gt;, such as “1 second desired; 2 seconds acceptable”</td>
</tr>
<tr>
<td>Measurable</td>
<td>&lt;&lt;Indicate how this goal can be measured with respect the specific elements it addresses – include as appropriate baseline measurements, minimum values, maximum values, average or typical or expected values, etc.&gt;&gt;, such as “time between hitting Enter and getting useful information on the screen”</td>
</tr>
<tr>
<td>Relevant</td>
<td>&lt;&lt;Describe system process (SSAD 2.3.1), mode (2.3.2), or artifact or information classifier (2.2) to which this level of service is relevant&gt;&gt;, such as “larger delays in order processing (Process–07 in SSAD 2.3.1) cause user frustration”</td>
</tr>
<tr>
<td>Specific</td>
<td>&lt;&lt;Describe what in particular within the system process (SSAD 2.3.1), mode (2.3.2), or artifact or information classifier (2.2) this level of service addresses&gt;&gt;, such as “credit card validation during order processing (Process–07 in SSAD 2.3.1) may cause significant delay when attempting to connect to the verification service”</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

1. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

2. L.O.S. goals specified here should be more specific than the L.O.S. Requirements (SSRD 5) or the system–level L.O.S. Goals (OCD 4.4).

**Common Pitfalls:**

- Including L.O.S. goals that do not reference L.O.S. Requirements (SSRD 5).
- Not allocating requirements to specific processes, modes, or information classifiers.
- Not satisfying the M.R.S. criteria.

**Model Integration Rules:**

- Each L.O.S. requirement (SSRD 5) should be described here.
- Each system–level L.O.S. Goals (OCD 4.4) should be described here.
- Each process listed here, should be described in the system’s Processes (SSAD 2.3.1).
- Each mode listed here, should be described in the system’s Modes of Operation (SSAD 2.3.2).
- Each artifact or information classifier listed here, should be described in the Artifacts & Information (SSAD 2.2) for the system.
2.4.1 **L.O.S. Goal X**

Create a section at this level for each L.O.S. requirement of the system that is the average or weighted average of the values for more than system element. The header of this section should be the name of the L.O.S. requirement and its unique designator, if you have assigned an identifier and it is different from the name. Create a L.O.S. Form (Table 11) to describe the goal for each system element to which the L.O.S. requirement applies.

### 2.5 Rules

Describe the rules that the system must implement in support of the policies (e.g. audit trails, information access, copyright protection) or constraints (inc. laws, regulations, and standards) that must be satisfied by the organization (OCD 4.5.4).

The goal is to document accurately important rules that affected the system’s design and implementation, without getting into implementation details.

**Representation:**

Create a list of system rules described in either informal text or a formal specification language (e.g. UML’s OCL). Clearly identify any system processes (SSAD 2.3.1), modes (SSAD 2.3.2), or artifact and information classifiers (SSAD 2.2) to which the rule is known to apply.

Create a table like that shown in Table 12 that shows how organization rules are supported by system rules and to which system elements the organization and system rules. If an organization rule is not supported by a system rule, put “N/A” (not applicable) in the System Rule column.

**Table 12: Map of Organization Rules to System Rules and Elements**

<table>
<thead>
<tr>
<th>Organization Rule Identifier</th>
<th>System Rule Identifier</th>
<th>Applies To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization Rule Identifier</td>
<td>System Rule Identifier</td>
<td>List of processes, modes, or artifact or information classifiers</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Trade–offs:**

See Trade–offs in section OCD 3.5(current organization rules).

**Recommendations:**

1. Describe each rule clearly and concisely.

2. To facilitate traceability, assign each rule a unique number (e.g. System-Rule-01 or SR-01).

**Common Pitfalls:**

- Describing a rule that is already captured in another view of the system (e.g. modes of operation (SSAD 2.3.2)).

- Describing an actors in a rule description and not describing the actor in the system context (SSAD 2.1).

- Describing an artifact or an information classifier in a rule description and not describing it in the system’s artifacts and information (SSAD 2.2).
Guidelines for MBASE System and Software Architecture Description (SSAD)

- Describing a process in a rule description and not describing it in the system’s processes (SSAD 2.3.1).
- Describing a mode in a rule description and not describing it in the system’s modes of operations (SSAD 2.3.2).

Model Integration Rules:

- Each organization rule described here, should be described in the proposed organization’s rules (OCD 4.5.4).
- Each actor described in a rule, should be defined in the system context (SSAD 2.1).
- Each artifact or an information classifier described in a rule, should be defined in the system’s artifacts and information (SSAD 2.2).
- Each process described in a rule, should be defined in the system’s processes (SSAD 2.3.1).
- Each mode described in a rule, should be defined in the system’s modes of operations (SSAD 2.3.2).
- LCA: Each organization rule (OCD 4.5.4) should be supported by one or more system rules.
- IOC: Each organization (OCD 4.5.4) & system rule should be implemented by one or more design elements described in Architecture Design & Analysis (SSAD 3) and/or Implementation Design (SSAD 4).

3. Architecture Design & Analysis

Analysis the problem and design a high–level, general architecture for the system that is independent of the implementation technology. Describe what are the work units (components) of the system, what the components are expected to do, how the components are connected, and what the components communicate.

The form of the component depends on the abstraction and architectural style employed. The following paragraphs characterize three categories of systems and their typical components.

- **System of systems** is a system that is a collection of loosely coupled systems, i.e. provide distinctly useful capabilities that can be used in various combinations or potentially alone. Each component is itself a system, whose architecture needs to be described unless reusing an existing system or purchasing a COTS or custom system. (An example of a System of Systems is a system that combines a navigation system, a vehicle–control system, a weapon–control system, and a life–support system.)

- **Composite system** is a system of separately executing components that are generally useful only when working with other components. At the architecture design level, the components are abstractions of the concurrent processes (also called “OS processes” or “tasks”), threads, databases, dynamic– or static–link libraries, CORBA or COM objects, or Enterprise Java Beans, which will be used in the implementation.

- **Simple system** is typically a system that is implemented as a single software executable running on a single computer or device. Sample software components include the software executable and its active objects and classes if using an object–oriented development; functions if using a function–oriented development, and rules if using a rule–oriented development.

**Representation**

Create a UML package with the stereotype <<systemModel>> the name “Architecture Design”. Create all models described in the following sections in this package.
Recommendations:

The subsections of this section described below are intended to be general purpose. Tailor this section based on the size of the system, the chosen representation language, and the architectural style(s).

Model Integration Rules:

- The rationale for the Architecture must be documented in the FRD.

Guidelines:

Using Rose, create a UML package with the stereotype <<systemModel>> the name “Architecture Design” in the top–level package with the name Logical View.

3.1 Structure

In the following subsections, describe the hardware and software components of the system, the actors for the system, any layering or partitioning of the components, and the connections among the components and actors. (In section 3.5, describe the architecture style(s), pattern(s), and framework(s) used, and any constraint that they impose.)

Notes:

Throughout Architecture Design & Analysis (SSAD 3) and Implementation Design (SSAD 4), the term component classifier is used for the kind of component (e.g. human), and the term component is used for the instance of component classifier (e.g. you'7).

Recommendations:

1. Taylor the subsections of this section based on the size of your system, the architecture language used, and in some cases, the architecture style(s) used. The subsection shown emphasize the hardware or software nature of components and connectors. This organization works fine for Simple Systems and some Composite Systems described in some architecture languages (and standard UML).

For other Composite Systems and for Systems of Systems, the document may be easier to follow and maintain if the subsections emphasized the topology described in subsection 3.1.1. For example, replace the subsections shown below with the subsections shown under 3.1.1.

If appropriate, the lowest–level topologically–based subsection may have subsections for hardware classifier, software classifier, hardware instances, software instances deployment models as shown bellow.

2. For Systems of Systems (or systems specified using the Society of Automotive Engineers’ (SAE) proposed Avionics Architecture Description Language (AAD)), combine the following subsections. (This recommendation can be combined with suggestion 1.)

- Merge the sections Hardware Classifier Model (SSAD 3.1.2) and Software Classifier Model (SSAD 3.1.3) into a single section Component Classifier Model.
- Merge the sections Hardware Component Classifiers (SSAD 3.1.5) and Software Component Classifiers (SSAD 3.1.7) into a single section Component Classifiers.

7 Because of our limited knowledge of the universe, the authors are assuming that if you are reading this document, then you are a human from the planet earth. No insult intended if you are not.
• Merge the sections Hardware Connector Classifiers (SSAD 3.1.6) and Software Connector Classifiers (SSAD 3.1.8) into a single section Connector Classifiers.

• Merge the sections Hardware Components (SSAD 3.1.9) and Software Components (SSAD 3.1.11) into a single section Components.

• Merge the sections Hardware Connectors (SSAD 3.1.10) and Software Connectors (SSAD 3.1.12) into a single section Connectors.

These subsections work better than the subsection shown below because most (if not all) components are systems, and each system can be a combination of hardware and software, just hardware, or just software.

UML Guideline:

Create a UML package with the stereotype <<Structure Model>> the name “Architecture”. Create all models described in the following sections in this package.

577 Guidelines:

Using Rose, create the UML package with the stereotype <<Structure Model>> the name “Architecture” in the <<systemModel>> package with the named “Architecture Design”. Create subpackages as described above for simple systems. (Note: do not use Rose’s Component View or Deployment View).

3.1.1 Topology

Describe the how the components and component classes of the system are organized, e.g. layers, partitions, and/or subsystems.

Systems with that have a relatively large number of components and component classes are commonly organized into combinations layers, partitions, and/or subsystems. The selection of layers, partitions, and/or subsystems depends on the architecture style(s) employed and the size of the system being described. The following paragraphs described common organization strategies for the three categories of systems described above.

• Simple Systems:

  Components are organized in two layers: Hardware and Software, where the Hardware layer contains all hardware components and component classifiers and the Software layer contains all software components and component classifiers.

• Composite Systems:

  Components may be grouped into subsystems that represent a behavior unit (e.g. the file management subsystem of an operating system).

  The components (or subsystems if they have been defined) are organized in two or more layers representing different levels of abstract functionality and grouping the components that implement the functionality. Typically the most abstract, complex, application–specific, or user–specific functionality is shown as the top layer; the hardware or the environment–specific functionality is shown at the bottom; and functionality that supports multiple applications and is not environment–specific is shown in the middle. The following are some common layer combinations.

  • Application, Business, System Software, Hardware
  • Application, Business, Middleware, System Software, Hardware

Where:
• the Application layer holds the application–specific services;
• the Business layer holds business–specific components that are used in several applications;
• the Middleware layer contains components such as GUI-builders, interfaces to database management systems, platform-independent operating system services, and OLE-components such as spreadsheets and diagram editors;
• the System Software layer holds components such as operating systems, databases, interfaces to specific hardware;
• the Hardware layer holds the hardware components. (Sometimes the System Software and Hardware layer are combined and called either Environment or Platform.)

• Application, Business Services, Business Objects, Middleware, System Software, Hardware
Like the previous two examples except that the Business layer is divided into separate layers for Services and Objects.

• Application, System Services, Resource Control, Hardware
Like the previous examples except that the System Software is divided into separate layers for Services and Resource Control and no common Business Services.

• Interface, Controller, Model
• Interface, Controller, Model, Persistence
(Often called “Model-View-Controller” layering) Where:
• the Interface layer holds the components used to connect this system to other systems or people;
• the Controller layer holds the components that provide services that are used by several interfaces and mediate access to the Model layer;
• the Model layer holds the components that support a common representation of the information needed by several interfaces;
• the Persistence layer holds the components that support the retention of the information needed between executions of the system;

In most layered architecture styles, components (or subsystems) in a can only request the services of components in a lower layer; although, components in a lower layer can send notices to components in a higher layer. Some styles impose strict layering, i.e. a component can only request the services of a component in the next lower layer and send notices to components in the next higher layer. Other styles are more relaxed, i.e. a component can request the services of a component in the any lower layer and send notices to components in the any higher layer.

Large layers are often subdivided into partitions. The following paragraphs describe some common criteria for partitioning a layer.

• Operational stakeholders (e.g. kind of users, maintainers, etc.) or what the stakeholders do with the system (e.g. edit documents, prepare spreadsheets). This principle is most commonly used for the upper layers.
Areas of developer competency. This principle is most commonly used for the middle and lower layers.

Security (e.g. partitioning based on who should have access).

Variability (e.g. separating components supporting optional features from required).

System of Systems:

Like the Composite System, the components, which are systems, are organized into two or more layers, which can be partitioned. In a System of System, each component is a system, so there is no need to group them into “subsystems”.

Representation:

Describe the topology of your system using:

- An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used.

- UML’s Static–Structure Diagrams. Create subpackages to organize your components. The following paragraphs described common organization strategies for the three categories of systems described above.

  - Simple Systems:

    Create one package with the stereotype and the name “Hardware Component Classifiers” that holds all hardware component and connector classifiers, and diagrams that illustrate their relations. Create a second package with the stereotype and the name “Software Component Classifiers” that holds all software components and connect classifiers, and diagrams that illustrate their relations. Create a third package which the name “Deployment Model” that holds hardware and software components (i.e. instances), and diagrams that show the configurations. Create Deployment Diagrams, Component Diagrams, as described in Hardware Classifier Model (SSAD 3.1.2), Software Classifier Model (SSAD 3.1.3), and Deployment Model (SSAD 3.1.4), to show the hardware classifiers, software classifiers, and configuration(s) of software and hardware components in the system.

  - Composite Systems:

    Create a Static–Structure Diagram that shows the layers or the subsystems used to organize the components.

    - Represent each subsystem as a package with the stereotype <<Subsystem>> and the name of the layer. If the components in a subsystem (“client”) require access to components in subsystem layer (“server”), show a dependency relation from the client subsystem to the serving subsystem.

      In the package representing each subsystem, create Deployment Diagrams, Component Diagrams, as described in Hardware Classifier Model (SSAD 3.1.2), Software Classifier Model (SSAD 3.1.3), and Deployment Model (SSAD 3.1.4), to show the hardware classifiers, software classifiers, and configuration(s) of software and hardware components in the subsystem.

    - Represent each layer as a package with the stereotype <<Layer>> and the name of the layer. If the components in a layer (“client”) require access to components in another layer (“server”), show a dependency relation from the client layer to the serving layer.

      In each package representing a layer,
• If the layer is not partitioned and the components are not organized into subsystems, create Deployment Diagrams, Component Diagrams, as described in Hardware Classifier Model (SSAD 3.1.2), Software Classifier Model (SSAD 3.1.3), and Deployment Model (SSAD 3.1.4), to show the hardware classifiers, software classifiers, and configuration(s) of software and hardware components in the layer.

• If the layer is not partitioned but the components are organized into subsystems, create a Static–Structure Diagram that shows the subsystems and their visibility relations as described above.

• If the layer is partitioned, create a Static–Structure Diagram that shows the partitions and their visibility relations. Represent each partition as a package with the stereotype <<Partition>>. If the components in a partition (“client”) require access to components in another partition (“server”), show a dependency relation from the client partition to the serving partition.

In each package representing a partition,

• If the components are not organized into subsystems, create Deployment Diagrams, Component Diagrams, as described in Hardware Classifier Model (SSAD 3.1.2), Software Classifier Model (SSAD 3.1.3), and Deployment Model (SSAD 3.1.4), to show the hardware classifiers, software classifiers, and configuration(s) of software and hardware components in the layer.

• If the components are organized into subsystems, create a Static–Structure Diagram that shows the subsystems and their visibility relations as described above.

• System of Systems:

  Create a Static–Structure Diagram that shows the layers used to organize the components

• Represent each layer as a package with the stereotype <<Layer>> and the name of the layer. If the components in a layer (“client”) require access to components in another layer (“server”), show a dependency relation from the client layer to the serving layer.

In each package representing a layer,

• If the layer is not partitioned, create Deployment Diagrams, Component Diagrams, as described in Hardware Classifier Model (SSAD 3.1.2), Software Classifier Model (SSAD 3.1.3), and Deployment Model (SSAD 3.1.4), to show the hardware classifiers, software classifiers, and configuration(s) of software and hardware components in the layer.

• If the layer is partitioned, create a Static–Structure Diagram that shows the partitions and their visibility relations. Represent each partition as a package with the stereotype <<Partition>>. If the components in a partition (“client”) require access to components in another partition (“server”), show a dependency relation from the client partition to the serving partition.

In each package representing a partition, create Deployment Diagrams, Component Diagrams, as described in Hardware Classifier Model (SSAD 3.1.2), Software Classifier Model (SSAD 3.1.3), and Deployment Model (SSAD 3.1.4), to show the hardware classifiers, software classifiers, and configuration(s) of software and hardware components in the layer.

(Ordinary packages can be also used to further organize components. Ordinary packages can be placed in a layer, partition, subsystem, or another package; but should not be mixed with layers, partitions, subsystems, or components. In each package that holds layer or ordinary packages, create a UML Static–Structure Diagram that shows the ordinary packages, and the dependency relations among them. For each package
(“client”) that requires the visibility to the contents of another package (“supplier”), draw a dependency arrow from the client package to the supplier.)

Describe each layer, partition, and subsystem in the subsections numbered 3.1.1.x, shown below. (You may have to adjust the paragraph numbers depending on the combination of layers, partitions, and subsystems that you are using.)

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

Common Pitfalls:

- Treating a component as a layer.
- Mixing layers, partitions, subsystems, and components on one UML diagram.  
- Forgetting a dependency relation between layers.
- Forgetting a dependency relation between partitions.
- Forgetting a dependency relation between subsystems.

Model Integration Rules:

- At LCO, draft layers, partitions, and subsystems should be identified.
- At LCA, 
  - Any layers and partitions shall be identified;
  - Any subsystems that are not implementation–specific shall be identified;

577 Guidelines:

577 projects are typically a cross between Simple and Composite Systems. The hardware is typically a simple networked environment that is not being built as part of the project. The software is typically a simple distributed application.

Create the three packages described for a Simple System. Layering the software component classifiers as described for Composite Systems may be useful; but, you are not likely to need partitions or subsystems.

(Note: do not use Rose’s Component View or Deployment View.)

3.1.1.1 Layer X

Create a section at this level for each layer described above. The header of this section should be the name of the layer and its unique designator, if you have assigned an identifier and it is different from the name. Describe the purpose of the layer. If the layer is partitioned or contains subsystems create subsections to describe them; otherwise, list the component classes and instances that make up this layer.
UML Guideline:

If the layer is partitioned or contains subsystems, then include a Static–Structure Diagram (as described above) that shows the partitions or shows the subsystems, and shows the dependency relations among them.

If layer contains components, then include references to the diagrams in the Hardware Classifier Model (SSAD 3.1.2), the Software Classifier Model (SSAD 3.1.3) and the Deployment Model (SSAD 3.1.4) that describe the components and classes that belong in this layer.

3.1.1.1.1 Partition X

Create a section at this level for each partition described above. The header of this section should be the name of the partition and its unique designator, if you have assigned an identifier and it is different from the name. Describe the purpose of the partition. If the contains subsystems create subsections to describe them; otherwise, list the component classes and instances that make up this partition.

UML Guideline:

If the partition contains subsystems, then include a Static–Structure Diagram (as described above) that shows the subsystems and the dependency relations among them.

If partition contains components, then include reference to diagrams in the Hardware Classifier Model (SSAD 3.1.2), the Software Classifier Model (SSAD 3.1.3) and the Deployment Model (SSAD 3.1.4) that describe the components and classes that belong in this partition.

3.1.1.1.1.1 Subsystem X

Create a section at this level for each subsystem described above. The header of this section should be the name of the subsystem and its unique designator, if you have assigned an identifier and it is different from the name. Describe the purpose of the subsystem. List the component classes and instances that make up this subsystem.

UML Guideline:

Include references to the diagrams in the Hardware Classifier Model (SSAD 3.1.2), the Software Classifier Model (SSAD 3.1.3) and the Deployment Model (SSAD 3.1.4) that describe the components and classes that belong in this subsystem.

3.1.2 Hardware Classifier Model

Describe the kinds of hardware components that are either part of the system or on which this system will run, the actors for the system with which the components interacts, and the kinds of connectors that will be used to connect them.

Representation:

Describe the hardware classes the system using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Deployment Diagrams. Create one or more UML Deployment Diagrams that shows the kinds of hardware as node classifiers with the name of the hardware type. If instances of two kinds of hardware can be connected (e.g. wire together), then connect their node classifiers with an association that has the stereotype <<connector>> and whose name is the class of the connector (e.g. “Bus”, “Network”).
Each Deployment Diagram should be accompanied by a brief description of its purpose.

(For many systems, a single Deployment Diagram is sufficient to represent the hardware classes.)

For each hardware component classifier, create a subsection numbered 3.1.5.x with the name of the hardware connector classifier in the header, under the Hardware Component Classifiers (SSAD 3.1.5). The body of the subsection should describe pertinent information about the kind of hardware.

For each hardware connector classifier, create a subsection numbered 3.1.6.x with the name of the hardware connector classifier in the header, under the Hardware Connector Classifiers (SSAD 3.1.6). The text of the subsection should describe the kind of hardware.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

**Trade-offs:**

ADL’s and UML both offer standardized semantics and require reviewers to know the notation or language. Some ADL’s offer semantics that support specialized analysis techniques (e.g. real–time scheduling analysis).

**Recommendations:**

1. Give the hardware component classifier a name that expresses the kind of hardware.
   - A good name is usually a noun (e.g. “Workstation”), a short noun phrase (e.g. “Static RAM”), or the noun form of a verb (e.g. “Server”).
   - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   - Clear, self-explanatory names may require several words (e.g. “UNIX Server” “Pentium IV”).

2. To facilitate traceability, assign each component a unique number (e.g. Component-10, HWCC-06).

**Common Pitfalls:**

- Confusing component instances with component classifiers.
- Including software component classifiers. (They should documented in SSAD 3.1.7)
- Including actors that are not described in the system context (SSAD 2.1).
- Including implementation component and connector classifiers or details, e.g. specific kinds of computer at LCO. (They should documented in SSAD 4.1.5 and 4.1.6)

**Model Integration Rules:**

- Each actor described here should be described the system context (SSAD 2.1).
- LCO: Describe classifiers of expected hardware components.
- LCA: complete descriptions of all hardware component classifier needed to support high–risk or architecturally–significant behaviors.
• IOC: at least one instance of each hardware component classifier shown here shall appear in the Deployment Model (SSAD 3.1.4).

577 Guidelines:

Rose does support the current UML semantics for the Deployment Diagram. Create one or more UML Static–Structure Diagrams that shows the node types represented as classifiers with the stereotype <<node>> and the name of the hardware kind, and shows the actors that interact with the hardware components. If instances of two hardware classifiers, or actor and a hardware classifier, can be connected (e.g. wire together), then connect their node classifiers with an association that has the stereotype <<connector>> and whose name is the class of the connector (e.g. “Bus”, “Network”).

3.1.3 Software Classifier Model

Describe the kinds of software components that are part of the system, the actors for the system with which the system interacts, and the kinds of connectors that will be used to connect them.

Representation:

Describe the software classes the system using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Component Diagrams. Create one or more UML Component Diagrams that shows the kinds of software components, their interfaces, their relations, and the classes and objects that reside on the component. Each kind of software components is represented as component classifiers with the name of the kinds of software components. Interfaces are represented as a circle (the stereotypical icon form) with an association connecting it to component that realizes it.

UML v1.4 [OMG 2001] does not define the concept of “connectors” for software components. If the instances of component classifier requires services from instances of another component classifier,

• If the interfaces have been defined (see Interface(s) (SSAD 3.1.7.1.2)) for the component classifier, then a dependency relation should be drawn from the component classifier that needs the service to the appropriate interface(s) of the supplying component classifier.

• Otherwise, a dependency relation should be drawn from the component that needs the service to the supplying component.

Each Component Diagram should be accompanied by a brief description of its purpose.

(For many systems, a single Component Diagram is sufficient to represent the software classes.)

For each software component classifier, create a subsection numbered 3.1.7.x with the name of the hardware connector classifier in the header, under the Software Component Classifiers (SSAD 3.1.7). The body of the subsection should describe pertinent information about the kind of software.

For each software connector classifier, create a subsection numbered 3.1.8.x with the name of the software connector classifier in the header, under the Software Connector Classifiers (SSAD 3.1.8). The text of the subsection should describe the kind of software connector.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).
Trade-offs:

ADL’s and UML both offer standardized semantics and require reviewers to know the notation or language. Some ADL’s offer semantics that support specialized analysis techniques (e.g. real–time scheduling analysis).

Recommendations:

1. Give each software component classifier a name that expresses the kind of software.
   - A good name is usually a noun (e.g. “Administrator”), a short noun phrase (e.g. “User Agent”), or the noun form of a verb (e.g. “Controller”), that is either a:
     - Common noun or noun phrase, i.e. the name of a class of persons, places, or things (e.g. “Apple”, “Building”, “F16”, “Computer”),
     - Mass or abstract noun or noun phrase, i.e. the name of a quality, activity, or substance (e.g. “water”, “traffic”, “software”, “cooking”, “usefulness”, “programming”, “completeness”),
     - Countable noun or noun phrase, i.e. the name of a thing that can be enumerated (e.g. “targets”, “code”) Either mass nouns or countable nouns may be modified by a unit of measure, i.e. means of referring to a quantity (e.g. “water volume in cubic centimeters”, “percent complete”, “number of targets”, “lines of code”).
   - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   - Clear, self-explanatory names may require several words (e.g. “Library User”, “System Administrator”).

2. To facilitate traceability, assign each component a unique number (e.g. Component-10, SWCC-05).

Common Pitfalls:

- Confusing component instances with component classifiers.
- Including hardware component classifiers. (They should documented in SSAD 3.1.2)
- Including actors that are not described in the system context (SSAD 2.1).
- Including implementation component classifiers or details, e.g. specific kinds of computer at LCO. (They should documented in SSAD 4.1)

Model Integration Rules:

- Each actor described here should be described the system context (SSAD 2.1).
- LCO: Describe classifiers of expected software components.
- LCA: complete descriptions of all software component classifier needed to support high–risk or architecturally–significant behaviors.
- IOC: at least one instance of each software component classifier shown here shall appear in the Deployment Model (SSAD 3.1.4).
577 Guidelines:

Rose does support the current UML semantics for the Component Diagram. Create one or more UML Static–Structure Diagrams that shows the node types represented as classifiers with the stereotype <<component>> and the name of the software kind. Interfaces are represented as a circle (the stereotypical icon form) with an association connecting it to component that realizes it.

If a component requires services from another component, whose interfaces have been defined (see Interface(s) (SSAD 3.1.7.1.2)), then a dependency relation should be drawn from the component that needs the service to the appropriate interface(s) of the supplying component. If a component requires services from another component whose interfaces for the supplying component have not been defined, then a dependency relation should be drawn from the component that needs the service to the supplying component.

(While UML v1.4 [OMG 2001] does not define the concept of “connectors” for software components, as described above, if you want to use the concept, you can connect two <<component>> classifiers with an association that has the stereotype <<connector>> and whose name is the class of the connectors (e.g. “adaptor”, “distributor”). The interface can be represented as the classifier for the role. If you represent connectors, then for each software connector classifier, create a subsection 3.1.8.x with the name of the software connector classifier in the header, under the Software Connector Classifiers (SSAD 3.1.8). The text of the subsection should describe the software connector.)

3.1.4 Deployment Model

Describe component and connector configuration(s) that make a working version of the system. (There may be more than one configurations, e.g. for different modes.) For each configuration, describe the instances of hardware and software component classes that participate in the configuration, the allocation of software components to the hardware components, the instances of hardware connector classes that link the hardware components, and the instances software connector classes that link the software components.

Representation:

Describe the deployment configurations using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Deployment Model. Create a package with the stereotype <<Deployment Model>> with the name “Configuration Name”, where “Configuration Name” is the name of the configuration (e.g. “Standard Configuration”, “High Assurance Configuration”, “In–flight Mode”) described by the diagrams in the package.

Create one or more UML Deployment Diagrams that shows hardware components, the software components that reside on each hardware component, the connectors between hardware components, and dependency relations between software components.

Each hardware components is represented as a node classifiers with a label of the form “instance_name : node_classifier_name”, “instance_name / role_name : node_classifier_name”, “/ role_name : node_classifier_name”, or “node_classifier_name” (e.g. “Local / User Station : Workstation”, “/Administrator: Workstation”, or “ISD Server : UNIX Server”).

Each software components is represented as a component classifiers with a label of the form “instance_name : node_classifier_name”, “instance_name / role_name : node_classifier_name”, “/ role_name : node_classifier_name”, or “node_classifier_name” (e.g. “Library_User”). The software components are drawn inside the hardware nodes.
If two hardware instances are connected (e.g. wire together), then connect their node classifiers with a link that has the stereotype <<connector>> with a label of the form “instance_name : node_classifier_name” or “: node_classifier_name” (e.g. “LAN : Network”).

If a software component requires services from another software component,

- If the interfaces have been defined (see Interface(s) (SSAD 3.1.7.1.2)) for the supplying component, then a dependency relation should be drawn from the component that needs the service to the appropriate interface(s) of the supplying component.
- Otherwise, a dependency relation should be drawn from the component that needs the service to the supplying component.

Each Deployment Diagram should be accompanied by a brief description of its purpose.

For each hardware component, create a subsection numbered 3.1.9.x with the name of the hardware component in the header, under the Hardware Components (SSAD 3.1.9). The body of the subsection should describe the hardware component.

For each hardware connector, create a subsection numbered 3.1.10.x with the name of the hardware connector in the header, under the Hardware Connectors (SSAD 3.1.10). The text of the subsection should describe the hardware connector.

For each software component, create a subsection numbered 3.1.11.x with the name of the software component in the header, under the Software Components (SSAD 3.1.11). The body of the subsection should describe the software component.

For each software connector, create a subsection numbered 3.1.12.x with the name of the software connector in the header, under the Software Connectors (SSAD 3.1.12). The text of the subsection should describe the software connector.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

**Recommendations:**

1. If the software component has a name, the name should describe an instance of hardware or software.

   - A good name is usually a noun, a short noun phrase, or the noun form of a verb, that is either a:
     - Proper noun or noun phrase, i.e. the name of a specific person, place, or thing (e.g. “UNIX”, “Internet”),
     - Direct reference, i.e. a reference to previously identified or known person, place, or thing without necessarily using name (e.g. “the user agent”, “the local workstation”, “the system administrator”).

   (Proper nouns are uncommon at this stage of development.)

2. If the software component has a role name, the name should describe the part played by the instance in the configuration (e.g. “the user workstation”, “the database server”).

   - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
• Clear, self-explanatory names may require several words (e.g. “Library User”, “System Administrator”).

3. To facilitate traceability, assign each component a unique number (e.g. Component-10, SWCI-05).

Common Pitfalls:

• Confusing component instances with component classifiers.

• Including implementation component classifiers or details, e.g. specific kinds of computer at LCO. (They should documented in SSAD 4.1)

Model Integration Rules:

• Each hardware component classifier described here should be shown in the Hardware Classifier Model (SSAD 3.1.2) and described in the section Hardware Component Classifiers (SSAD 3.1.5).

• Each hardware connector classifier described here should be shown in the Hardware Classifier Model (SSAD 3.1.2) and described in the section Hardware Connector Classifiers (SSAD 3.1.6).

• Each software component classifier described here should be shown in the Software Classifier Model (SSAD 3.1.3) and described in the section Software Component Classifiers (SSAD 3.1.7).

• Each software connector classifier described here should be shown in the Software Classifier Model (SSAD 3.1.3) and described in the section Software Connector Classifiers (SSAD 3.1.8).

• Each actor described here should be described the system context (SSAD 2.1).

• At LCO, a System Deployment Model should show the expected configuration of hardware and software components, whose classifiers are described in the Hardware Classifier Model (SSAD 3.1.2) and the Software Classifier Model (SSAD 3.1.3).

• At LCA, the System Deployment Model shall show the final configuration of hardware and software that participate in high–risk behaviors, and the preliminary allocation of all other components.

• At IOC, the System Deployment Model shall show the final configuration of all components.

577 Guidelines:

Rose does support the current UML semantics for the Deployment Diagram. Create an UML Object Diagram (a UML Static–Structure Diagram with instances and no classifiers) within the “Deployment Model” package that shows the hardware instances and the hardware connection instances. Each node is represented as classifiers with the stereotype <<node>> and a label of the form “instance_name : node_classifier_name”, “instance_name / role_nme : node_classifier_name”, “/ role_name : node_classifier_name”, or “: node_classifier_name”.

For each node instance that has a unique configuration of hardware or software residing on it, create a sub–Package of the “Deployment Model” package with the stereotype <<node>> and a label of the form “instance_name : node_classifier_name”, “instance_name / role_nme : node_classifier_name”, “/ role_name : node_classifier_name”, or “: node_classifier_name”.

In each node package, create an UML Object Diagram that shows the component instances that reside on that node and the component instances that the resident components use. Each component instance is represented as a classifier with the stereotype <<component>> and a label of the form “instance_name : node_classifier_name”, “instance_name / role_nme : node_classifier_name”, “/ role_name : node_classifier_name”, or “: node_classifier_name”. If the instance resides on this node, then the instance_name should be the simple name of the component; otherwise, the qualified name of the component. For each component, show the following.
a. The interfaces of the component that have been identified. (At LCA all component interfaces should be identified.)

b. For each interface of the component, a realize relation from the component to its interface.

c. For each component used, a uses relation to the interfaces of the used component if its interfaces have been identified, or to the used component if its interfaces have not been identified.

### 3.1.5 Hardware Component Classifiers

For each hardware component classifier shown in the figure(s) shown in the Hardware Classifier Model (SSAD 3.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware component classifier in the header.

**Representation:**

The description of hardware components classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations. For example, if you are describing a system where the hardware is not being developed as part of the system (e.g. a web application), it is usually sufficient to just describe the purpose of each hardware component and the L.O.S. that the hardware is expected to provide.

**577 Guidelines:**

Your system is probably simple enough, that you need only fill in the following sections for each component.

- 3.1.5.1.1 Purpose
- 3.1.5.1.5 L.O.S. Goals (needed to determine some system goals)

It is probably less risky to either defer describing the hardware in more detail until Implementation Design (SSAD 4) or not describe the hardware interfaces at all then to describe this detail here.

If you decide more detail is needed, create a UML package, with the stereotype <<node>> and the name “Component Classifier Name AD”, in the package that holds the classifier that represents this hardware component classifier.

### 3.1.5.1 Component Classifier X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

#### 3.1.5.1.1 Purpose

Describe the purpose of this class of component.

#### 3.1.5.1.2 Interface(s)

Describe the visible features of the component classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

**Representation:**

Describe the visible features of the component classifier using either:
a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Static–Structure Diagrams & Deployment Diagrams. A set of services is called an interface, which is represented as a classifier with the stereotype <<interface>> and the name of the set (e.g. “Security Services” or “File Management”). Note: an interface in UML is not allowed to have either attributes or parts.

Create a UML Static–Structure Diagram named “Interface Details” that defines the interfaces of the hardware component classifier, the operations they contained, and any inheritance relations among them. (Note: these interface classifiers may be used to define multiple components, connectors, and classes.)

Create a UML Deployment Diagram named “Interfaces” that shows the node classifiers for the component being described and its visible subcomponents; the interface classifiers (they will appear in icon form) define in the Static–Structure Diagram, and its visible attributes. Connect the node representing the component being described to each interface with an association. To represent a visible subcomponent to the component being described, either (a) connect the node representing the component being described using a composition relation to each node that represents a subcomponent; or (b) graphically nest each node classifier representing a subcomponent in the node classifier representing the component being described. Add any attributes directly to the component being described. Set the visibility of each subcomponent and each attribute to public. (Operations in interfaces are always public.)

Note: the component’s Interface(s) (SSAD 3.1.5.1.2) and Parameters (SSAD 3.1.5.1.3) can be represented using one Deployment Diagram. (Name the diagram “Interfaces & Parameters”.)

For each feature or set of features, create a subsection numbered 3.1.5.1.2.x and the name of the hardware feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Model Integration Rules:

- Each interface feature described here, should be used in the component’s behavior description (SSAD 3.1.7.1.4).
- At LCA: a preliminary set of interfaces for the component shall be described.
- At IOC: the final set of interfaces for the component shall be described.

577 Guidelines:

Rose does support the current UML semantics for the Deployment Diagram, including much of what is described above. However, your system is probably simple enough, that you are unlikely to need this detailed of a description for your hardware. (It is probably less risky to defer describing hardware interfaces until Implementation Design (SSAD 4) or to not describe the hardware interfaces at all.)

If you decide this level of detail is needed, create the Static–Structure Diagram named “Interfaces”.

- Add a classifier with the stereotype <<node>> for the component being described and for each of its visible subcomponents.
- Add a classifier with the stereotype <<interface>> for each set of related operations. Define the operations they contained and show any inheritance relations among the interfaces.
- Connect the <<node>> classifier representing the component being described to each interface with a realization relation.
- Add any attributes of the hardware component to the <<node>> classifier.
• Add a classifier with the stereotype <<node>> for each of its visible subcomponents.

• Add a composition relation from the <<node>> classifier representing the component being described to each <<node>> classifier that represents a subcomponent. (If the <<node>> classifier representing the component being described does not have any attributes, then graphically nest each <<node>> classifier that represents a subcomponent inside the node representing the component being described can be an alternate representation for composition; but many tools do not recognize the relation.)

• Set the visibility of each subcomponent and each attribute to public. (Operations in interfaces are always public.)

Note: the component’s Interface(s) (SSAD 3.1.5.1.2) and Parameters (SSAD 3.1.5.1.3) can be represented using one Static–Structure Diagram. (Name the diagram “Interfaces & Parameters”.)

3.1.5.1.2.1 Feature or Feature Set X

Create a section at this level for each feature or feature set described above. The header of this section should be the name of the feature or feature set and its unique designator, if you have assigned an identifier and it is different from the name. The contents of this section should include a description of the feature’s purpose and the following:

• If the feature is an attribute, include its classifier and any default value;

• If the feature is a node that is a part, include a reference to its description in the Internal Architecture (SSAD 3.1.5.1.7) section;

• If the feature is an operation, include a description of its parameters and result (called a signature), and any pre- and post-condition;

• If describing a feature set, describe the each member of the set as defined in the previous three bullets.

3.1.5.1.3 Parameters

Describe any parameters of the hardware component classifier. The parameters need to be set when an instance of the hardware component classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other components.

Representation:

Describe the parameters of the component classifier using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Deployment Diagrams. Create a UML Deployment Diagram that shows the node classifier for the component being described. Add a dashed rectangular box to upper–right hand corner of the node classifier. List the parameters in the box.

Note: the component’s Interface(s) (SSAD 3.1.5.1.2) and Parameters (SSAD 3.1.5.1.3) can be represented using one Component Diagram.

Create a table like Table 13 that lists each parameter.
Table 13: Parameter Table for Classifiers

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type/Signature</th>
<th>Default Value</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name &amp; Identifier</td>
<td>(see below)</td>
<td>What to use if no argument is bound to this parameter</td>
<td>A brief description of the purpose of the parameter.</td>
</tr>
</tbody>
</table>

The Type/Signature field depends on the kind of parameter being described. The following paragraphs describe some common contents for the Type/Signature field:

- If the parameter is an instance (e.g., an object, a component), then the contents must be the name of a classifier. If a value is expected, put “in” before the classifier name.
- If the parameter is an operation, then the contents is the operations signature.
- If the parameter is a classifier, then the contents are either blank (as in UML) or should say “Classifier”.

(Note: the “kind” of the parameter is determined by the contents of the Type/Signature field.)

Model Integration Rules:

- At LCA: a preliminary set of parameters for the component, if any exist, shall be described.
- At IOC: the final set of parameters for the component shall be described. (If none, say so.)

577 Guidelines:

Rose does not fully support the current UML semantics for the Deployment Diagram, including what is described above. However, your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware. (It is probably less risk to defer describing hardware parameters until Implementation Design (SSAD 4) or to not describe the hardware parameters at all.)

If you decide this level of detail is needed, create a Static–Structure Diagram instead of the UML Deployment Diagram described above. In the Static–Structure Diagram, create a classifier with the stereotype <<node>> for the component being described. Add a dashed rectangular box to upper–right hand corner of the classifier. List the parameters in the box.

Note: the component’s Interface(s) (SSAD 3.1.5.1.2) and Parameters (SSAD 3.1.5.1.3) can be represented using one Component Diagram.

3.1.5.1.4 Behavior

Describe behavior of the instances of this component classifier.

Representation:

The representation of behavior is depends on the language used to represent it. The following subsection work for UML–based architecture descriptions and may work for ADL’s.

Model Integration Rules:

- At LCA: a draft behavior for the component shall be described if the component is high–risk.
- At IOC: the behavior for the component shall be described if the component is included in the build.
UML Guidelines:

Describe the processes of this component classifier, including how it works with the other components and the system’s actors, and how it uses the artifacts and information; and if significant, how the component’s behavior depends on its mode.

577 Guidelines:

Your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware. (It is probably less risk to defer describing hardware behavior until Implementation Design (SSAD 4) or to not describe the hardware behavior at all.)

3.1.5.1.4.1 Processes

Describe the processes of this component classifier. For each process, identify which other components and actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by this component and each actor during the process.

Representation:

Create a Use–Case Model that describes the component’s processes in the same way that a model was created to describe the system’s processes in SSAD 4.3. Represent the model as a package with the stereotype <<use–case model>> with the package name “Component Classifier Name Processes”.

- Create one or more Use–Case Diagrams
- For each process, create a Use–Case Description, and an Activity Diagram.

For each process, create a subsection numbered 3.1.5.1.4.1.x (see example section below) with the name of the process in the header. Each subsection should provide the information shown in the Figure 4 and an Activity Diagram.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

See Recommendations in 4.3.

1. Use the “just do it” approach to eliminate the pressure to get go right set of processes on the first pass (like writing a rough draft for a term paper). Start with the component’s actions used in the system’s processes that are associated with each capability (see the “Used In” field of the capabilities description). Each process may require several iterations to get right. “Go with what you know”, then revise the set of processes and adjust descriptions, as needed.

2. To facilitate traceability, assign each process a unique number (e.g. Process-01 or UC-01).

3. For LCA

- The following fields of Use–Case Description should be filled out for all use–cases: Name, Purpose, Actors, Importance, Requirements, Development Status, Pre–conditions, Post–conditions, and Includes.

- Typical, Alternate, and Exceptional Courses of Actions should be described for high–risk, architecturally significant, or particularly complex use–case. Fill–in the Includes & Extension Points fields if the courses of actions specify the inclusion of other use–cases or extension point, respectively.
4. For IOC, use–cases descriptions should be completely filled out for all use–cases.

**Common Pitfalls:**

- See Common Pitfalls in 4.3.
- Spending too much time describing the processes in detailed for low risk components. This problem is more common for Simple Systems and small Composite Systems. (Defer details until Implementation Design (SSAD 4.).)

**Model Integration Rules:**

- Each operation defined in the component’s interface (SSAD 3.1.5.1.2) must either be represented by a use–case or represented as an action in a use–case’s Activity Diagram.
- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).
- Each component in a process description should be defined in the architecture of the system (SSAD 3.1).
- Each classifier used or produced in a process description should be defined in the analysis classes for the system’s architecture (SSAD 3.2).

### 3.1.5.1.4.1.1 Process X

Create a section at this level for each process of the component. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.

### 3.1.5.1.4.2 Modes of Operation

Describe the modal behavior of the component instances, i.e. how the behavior of the component depends on the mode it is in. Describe the high–level states of the component, the events that cause the component to change modes, and how the processing is different in each mode.

**Representation:**

Create a State Model that describes the component’s modes. Represent the model as a package with the stereotype <<state model>> with the package name “Modes of Component Classifier Name”. Create one or more Statecharts that show the modes of component and the events that cause mode changes in the same way that a Statechart was created to describe the system’s modes in SSAD 2.3.2.

For each mode, create a subsection numbered 3.1.5.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list component’s processes (SSAD 3.1.5.1.4.1) are available in that mode (see Table 14). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode–specific behavior).

<table>
<thead>
<tr>
<th>Table 14: Component Processes Available in Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes</td>
</tr>
<tr>
<td>Process id &amp; name</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).
577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right-click on the <<node>> classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Modelx” that is attached to the <<node>> classifier, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

Recommendations:

See Recommendations for system’s Modes of Operation (SSAD 2.3.2).

Common Pitfalls:

- Spending too much time describing the modes in detailed for low-risk components or components that do not have significant modal behavior. This problem is more common for Simple Systems and small Composite Systems. (Defer details until Implementation Design (SSAD 4).)

- For additional pitfalls, see Common Pitfalls for system’s Modes of Operation (SSAD 2.3.2).

Model Integration Rules:

- Each component process (SSAD 3.1.5.1.4.1) should be listed as available in one or more modes.

- Each process listed as available in one or more modes should appear in described in SSAD 3.1.5.1.4.1.

3.1.5.1.4.2.1 Mode X

Create a section at this level for each mode of the component. The header of this section should be the name of the mode and its unique designator, if you have assigned an identifier and it is different from the name. Describe the mode (e.g. for the airplane’s taxi mode, “when in this mode, the airplane is moving on the ground, usually to or from a gate”).

List the component’s processes (SSAD 3.1.5.1.4.1) that are available in this mode (see Table 14). For each process, describe any effect of the mode

3.1.5.1.5 L.O.S. Goals

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component classifier. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a component classifier defines the goals for all instances of this classifier, and the goals that an implementation of this component classifier must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the system’s goals (i.e. the system goals will be meet its goals if all connectors and components achieve their goals).

Representation:

For each system goal that applies to this component class fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its own value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).
Guidelines for MBASE: System and Software Architecture Description (SSAD)

Recommendations:

1. Consider the system processes in which instances of this component classifier participates, when determining which system L.O.S. goals apply to this component classifier. L.O.S. goals that apply to the system processes (SSAD 2.4) generally apply to the components that participate in the process.

2. Consider derived goals, i.e. a system–level throughput requirement may mean imply a maximum time to complete one of this component classifier’s processes.

3. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

Common Pitfalls:

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.

Model Integration Rules:

- Each system L.O.S. Goals (OCD 4.4) should be described here.
- Each process listed here, should be described in the component’s Processes (SSAD 3.1.5.1.4.1).
- Each mode listed here, should be described in the system’s Modes of Operation (SSAD 3.1.5.1.4.2).
- Each artifact or information classifier listed here, should be described in the architecture’s Analysis Classes (SSAD 3.2).

3.1.5.1.6 Constraints

Describe any constraints on the use and implementation of this component that are not captured in other sections of the component description. The following paragraphs describe some typical constraints.

- Rules that the component must implement in support of rules that the system must satisfy;
- Constraints imposed on component’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on component’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the component’s design and implementation, without getting into implementation details.

Representation:

Create a list of system rules described in either informal text or a formal specification language (e.g. UML’s OCL). Clearly identify any system processes (SSAD 2.3.1), modes (SSAD 2.3.2), or artifact and information classifiers (SSAD 2.2) to which the rule is known to apply.

Create a table like that shown in Table 15 that shows how organization rules are supported by system rules and to which system elements the organization and system rules. If an organization rule is not supported by a system rule, put “N/A” (not applicable) in the System Rule column.
Table 15: Map of Component Constraints to Sources and Elements

<table>
<thead>
<tr>
<th>Trace's Back To</th>
<th>Component Rule</th>
<th>Applies To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier of System Rule or Architecture Style, Pattern Or Framework</td>
<td>Component Rule Identifier</td>
<td>List of component’s processes, modes, or artifact or information classifiers</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD).

**Trade–offs:**

See Trade–offs in section OCD 3.5(current organization rules).

**Recommendations:**

1. To facilitate traceability, assign each rule a unique number (e.g. Component-Rule-01 or CR-01).

2. For additional recommendations, see Recommendations in section OCD 3.5(current organization rules).

**Common Pitfalls:**

- Describing a rule that is already captured in another view of the component (e.g. modes of operation (SSAD 3.1.5.1.4.2)).

**Model Integration Rules:**

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each process described in a constraint, should be described in the component classifier’s Processes (SSAD 3.1.5.1.4.1).
- Each mode described in a constraint, should be described in the component classifier’s Modes of Operation (SSAD 3.1.5.1.4.2).
- Each artifact or information classifier described in a constraint, should be described in the architecture’s Analysis Classes (SSAD 3.2).
- IOC: Each component rule should be implemented by one or more elements described in either this component’s Internal Architecture (SSAD 3.1.5.1.7), or its Implementation Design (SSAD 4).

3.1.5.1.7 **Internal Architecture**

Describe the architecture of this hardware component classifier that is independent of the implementation technology. Describe the subcomponents of this component, what they are expected to do, how they are connected, and what they communicate.

**Representation:**

Apply the same guidelines to describe the architecture of a component as was used to describe for the system architecture in section Architecture Design & Analysis (SSAD 3), i.e. create subsections: “Structure”, “Analysis Classes”, “Behavior”, “L.O.S., Projected”, and “Architectural Styles, Patterns & Frameworks”. 
Guidelines for MBASE  
System and Software Architecture Description (SSAD)

Recommendations:

1. For a System of System, just include a reference to the project documents (e.g. OCD, SSAD) for the project that is responsible for creating the subsystem represented by this component.

2. Only specify the internal architecture for a hardware component when the component will to be implemented by the project or when the internal architecture is needed to analyze the system. For a Small System or Composite System, the internal architectures of hardware components are often not important.

3. You need not specify the architecture of a component that according to your architecture language or style is the lowest level that can be specified (i.e. “atomic”\(^9\)). A complete architecture specification would define an internal architecture for all but the atomic components. However, schedule or cost constraints may stop the architecture design process before uniformly reaching all atomic components. A project may also stop if it determines that there is little benefit from specifying all atomic components.

Common Pitfalls:

- Spending time describing the internal architecture for low risk components, including components that will not be implemented by this project. (This problem is more common for Simple Systems and small Composite Systems.) If the component is low risk, consider deferring its details until Implementation Design (SSAD 4).

577 Guidelines:

While your system is probably a Composite System, it is probably simple enough, that you are unlikely to need this detailed of a description for your hardware.

If you decide the node’s internal architecture needs to be specified, create a UML package with the stereotype \(<\text{Structure Model}>\) the name “Architecture” in the package representing this component (i.e. the one with stereotype \(<\text{node}>\) and the name “\(\text{Component Classifier Name AD}^\)”) to hold the internal architecture description.

3.1.6 Hardware Connector Classifiers

For each hardware connector classifier defined in the figure(s) shown in the Hardware Classifier Model (SSAD 3.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware connector classifier in the header.

Representation:

The description of hardware connector classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations. For example, if you are describing a system where the hardware is not being developed as part of the system (e.g. a web application), it is usually sufficient to just describe the purpose of each hardware connector and the L.O.S. that the hardware is expected to provide.

The following subsections are intended to support a wide range of systems, architecture languages, and architecture styles.

---

\(^9\) Calling a component “atomic” means that the component may have an internal structure; but it is not important that for the abstraction defined by the ADL. The component can be treated as if it has no internal structure for some purpose of this specification. This use of the word “atomic” is consistent with its use in Chemistry, where we know that each atom is composed of electrons, protons, and neutrons; but they are unimportant for some forms of chemical analysis.
577 Guidelines:

Your system is probably simple enough, that you need only fill in the following sections for each connector.

- 3.1.6.1.1 Purpose
- 3.1.6.1.5 L.O.S. Goals (needed to determine some system goals)

It is probably less risky to either defer describing the hardware in more detail until Implementation Design (SSAD 4) or not describe the hardware interfaces at all.

If you decide this level of detail is needed, create a UML package with the stereotype «connector>> the name “Connector Classifier Name AD” in the package that holds the corresponding association with the stereotype «connector>>.

3.1.6.1 Connector Classifier X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

3.1.6.1.1 Purpose

Describe the purpose of this class of connector.

3.1.6.1.2 Interface(s)

Describe the visible features of the connector classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

Representation:

Describe the visible features of the connector classifier using an Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

For each feature or set of features, create a subsection numbered 3.1.6.1.2.x and the name of the hardware feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

UML Guidelines:

UML does not describe how to represent the interface of connectors. (It says little about hardware connectors.) However, if your UML tools gives a UML node of a classifier as described in the semantics section of the UML Specification [OMG 2001], you can use a combination of a Static–Structure Diagram & Deployment Diagram. If not, you can use just a Static–Structure Diagram.

a. Static–Structure Diagram & Deployment Diagram Representation

Create a UML Static–Structure Diagram named “Interface Details” that defines the interfaces (see 3.1.5.1.2 for a description) of the hardware connector classifier, the operations they contained, and the inheritance relations among them. (Note: these interface classifiers may be used to define multiple connectors, components, and classes without redefining the interface classifier.)
Create a UML Deployment Diagram named “Interfaces”. Represent the connector as an association between the node classifiers it connects. Add to the connector an “association class” that has the same name and stereotype as the connector. Add the interface classifiers define in the Static–Structure Diagram. (The interfaces will appear in icon form.) Show an association between the connector’s association class and each interface.

Add a node classifier with the name of the subcomponent for each visible hardware subcomponent. Either show using a composition relation from the connector’s association class to each node that represents a subcomponent; or graphically nest each node classifier that represents a subcomponent inside connector’s association class. Add any attributes directly to connector’s association class. Set the visibility of each subcomponent and each attribute to public. (Operations in interfaces are always public.)

Note: the Connector’s Interface(s) (SSAD 3.1.6.1.2) and Parameters (SSAD 3.1.6.1.3) can be represented using one Deployment Diagram. (Name the diagram “Interfaces & Parameters”.)

b. Static–Structure Diagram Representation

Create the Static–Structure Diagram named “Interfaces”. Represent the connector as a classifier with the stereotype <<connector>> that has the name of the connector. Represent each as a classifier with the stereotype <<interface>>. Define the operations in each interface and show any inheritance relations among the interfaces. Show a realization relation between the <<connector>> classifier and each interface.

If the connector has any visible attributes, then add them to the <<connector>> classifier.

If the connector has any visible subcomponents, add a classifier with the stereotype <<node>> for each of its visible hardware subcomponents. Show a composition relation from the <<connector>> classifier to each <<node>> classifier that represents a subcomponent. (If the <<connector>> classifier does not have any attributes, then graphically nest each <<node>> classifier that represents a subcomponent inside the node representing the component being described can be an alternate representation for composition; but many tools do not recognize the relation.)

Set the visibility of each subcomponent and each attribute to public. (Operations in interfaces are always public.)

Note: the Connector’s Interface(s) (SSAD 3.1.6.1.2) and Parameters (SSAD 3.1.6.1.3) can be represented using one Static–Structure Diagram. (Name the diagram “Interfaces & Parameters”.)

577 Guidelines:

Your system is probably simple enough, that you are unlikely to need to describe the interfaces of hardware connector classifiers. If you decide this level of detail is needed, use the Static–Structure Diagram Representation described in the UML Guidelines.

3.1.6.1.2.1 Feature or Feature Set X

Create a section at this level for each feature or feature set described above. The header of this section should be the name of the feature or feature set and its unique designator, if you have assigned an identifier and it is different from the name. The contents of this section should include a description of the feature’s purpose and the following:

- If the feature is a attribute, include its classifier and any default value;
- If the feature is a node that is a part, include a reference to its description in the Internal Architecture (SSAD 3.1.5.1.7) section;
- If the feature is an operation, include a description of its parameters and result (called a signature), and any pre– and post–condition;
If describing a feature set, describe the each member of the set as defined in the previous three bullets.

### 3.1.6.1.3 Parameters

Describe any parameters of the hardware connector classifier. The parameters need to be set when an instance of the connector classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other connectors.

**Representation:**

Describe the parameters of the connector classifier using an Architecture Description Language (ADL). The form of the parameter specification will depend on the language used. (See the ADL’s manual for details.)

Create a table like Table 13 (on page 6) that lists each parameter.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Model Integration Rules:**

- At LCA: a preliminary set of parameters for the connector, if any, shall be described.
- At IOC: the final set of parameters for the connector, if any, shall be described.

**UML Guidelines:**

UML does not describe how to represent the parameters of connectors. (It says little about hardware connectors.) However, if your UML tools gives a UML node of a classifier as described in the semantics section of the UML Specification [OMG 2001], you can use a Deployment Diagram. If not, you can use just a Static–Structure Diagram.

- **Deployment Diagram Representation**
  
  Create a Deployment Diagram named “Parameters”. Represent the connector as an association between the node classifiers it connects. Add to the connector an “association class” that has the same name and stereotype as the connector. Add a dashed rectangular box to upper–right hand corner of the connector’s association class. List the parameters in the box.

- **Static–Structure Representation**
  
  Create a Static–Structure Diagram named “Parameters”. Represent the connector as a classifier with the stereotype <<connector>>. Add a dashed rectangular box to upper–right hand corner of the classifier. List the parameters in the box.

Note: the connector’s Interface(s) (SSAD 3.1.6.1.2) and Parameters (SSAD 3.1.6.1.3) can be represented using one Connector Diagram.

**577 Guidelines:**

Rose does support the current UML semantics for the Deployment Diagram, including what is described above. However, your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware. If you decide this level of detail is needed, follow the UML Guidelines described above.

### 3.1.6.1.4 Behavior

Describe behavior of the instances of this connector classifier.
Guidelines for MBASE System and Software Architecture Description (SSAD)

**Representation:**

Describe the behavior of the connector classifier using an Architecture Description Language (ADL). The form of the specification will depend on the language used. (See the ADL’s manual for details.)

The following subsection work for UML–based architecture descriptions and may work for other ADL’s.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

For systems where the hardware is not being developed as part of the project, it is usually unnecessary to specify its behavior.

**Model Integration Rules:**

- At LCA: a draft behavior for the connector shall be described if the connector is high–risk.
- At IOC: the behavior for the connector shall be described if the connector is included in the build.

**UML Guidelines:**

Describe the processes of this connector classifier, including how it works with the other connectors and the components; and if significant, how the connector’s behavior depends on its mode.

**577 Guidelines:**

Your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware. (It is probably less risk to defer describing hardware behavior until Implementation Design (SSAD 4) or to not describe the hardware behavior at all.)

**3.1.6.1.4.1 Processes**

Describe the processes of this connector classifier. For each process, identify which other connectors and actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by this connector and each actor during the process.

**Representation:**

As in 4.3, create a Use–Case Model that describes the connector’s processes. Represent the model as a package with the stereotype <<use–case model>> with the package name “Connector Classifier Name Processes”.

- Create one or more Use–Case Diagrams
- For each process, create a Use–Case Description, and an Activity Diagram.

For each process, create a subsection numbered 3.1.6.1.4.1.x (see example section below) with the name of the process in the header. Each subsection should provide the information shown in the Figure 4 and an Activity Diagram.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

See Recommendations in 4.3.
Guidelines for MBASE System and Software Architecture Description (SSAD)

1. Use the “just do it” approach to eliminate the pressure to get right set of processes on the first pass (like writing a rough draft for a term paper). Start with the connector’s actions used in the system’s processes that are associated with each capability (see the “Used In” field of the capabilities description). Each process may require several iterations to get right. “Go with what you know”, then revise the set of processes and adjust descriptions, as needed.

2. To facilitate traceability, assign each process a unique number (e.g. Process-01 or UC-01).

3. For LCA
   • The following fields of Use–Case Description should be filled out for all use–cases: Name, Purpose, Actors, Importance, Requirements, Development Status, Pre–conditions, Post–conditions, and Includes.
   • Typical, Alternate, and Exceptional Courses of Actions should be described for high–risk, architecturally significant, or particularly complex use–case. Fill–in the Includes & Extension Points fields if the courses of actions specify the inclusion of other use–cases or extension point, respectively.

4. For IOC, use–cases descriptions should be completely filled out for all use–cases.

Common Pitfalls:
See Common Pitfalls for system’s System Capabilities (SSAD 4.3)

Model Integration Rules:
• Each operation defined in the connector’s interface (SSAD 3.1.6.1.2) must either be represented by a use–case or represented as an action in a use–case’s Activity Diagram.
• Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).
• Each component in a process description should be defined in the architecture of the system (SSAD 3.1).
• Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information for the system’s architecture (SSAD 3.2).

3.1.6.1.4.1.1 Process X
Create a section at this level for each process of the connector. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.

3.1.6.1.4.2 Modes of Operation
Describe the modal behavior of the connector instances, i.e. how the behavior of the connector depends on the mode it is in. Describe the high–level states of the connector, the events that cause the connector to change modes, and how the processing is different in each mode.

Representation:
Create a State Model that describes the connector’s modes in the same way that a model was created to describe the system’s modes in SSAD 2.3.2. Represent the model as a package with the stereotype <<state model>> with the package name “Modes of Connector Classifier Name”. Create one or more Statecharts that show the modes of connector and the events that cause mode changes.
For each mode, create a subsection numbered 3.1.6.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list connector’s processes (SSAD 3.1.6.1.4.1) are available in that mode (see Table 16). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode–specific behavior).

### Table 16: Connector Processes Available in Mode

<table>
<thead>
<tr>
<th>Processes</th>
<th>Mode Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process id &amp; name</td>
<td>Describe any mode–specific behavior or L.O.S. goals of the process</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**577 Guidelines:**

In Rational Rose, instead of creating a separate package for the State Model, right–click on the connector’s association class, which you should have created when you described the connector’s interfaces (SSAD 3.1.6.1.2),10 in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Model” that is attached to the connector’s association class, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

**Recommendations:**

See Recommendations for system’s Modes of Operation (SSAD 2.3.2).

**Common Pitfalls:**

See Common Pitfalls for system’s Modes of Operation (SSAD 2.3.2).

**Model Integration Rules:**

- Each connector process (SSAD 3.1.6.1.4.1) should be listed as available in one or more modes.
- Each process listed as available in one or more modes should appear in described in SSAD 3.1.6.1.4.1.

### 3.1.6.1.4.2.1 Mode X

Create a section at this level for each mode of the connector. The header of this section should be the name of the mode and its unique designator, if you have assigned an identifier and it is different from the name. Describe the mode (e.g. for the airplane’s taxi mode, “when in this mode, the airplane is moving on the ground, usually to or from a gate”).

List the connector’s processes (SSAD 3.1.6.1.4.1) that are available in this mode (see Table 16). For each process, describe any effect of the mode

### 3.1.6.1.5 L.O.S. Goals

Describe how the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be **measurable, relevant, and specific** (M.R.S.).

The L.O.S. Goals for a connector classifier defines the goals for all instances of this classifier, and the goals that an implementation of this connector classifier must achieve. The classifier’s goals are used to demonstrate that the

---

10 If you are describing the behavior of a connector, then you should have described one or more interfaces.
architecture will meet the system’s goals (i.e. the system goals will be meet its goals if all connectors and components achieve their goals).

**Representation:**

For each system goal that applies to this connector fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its on value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

**Recommendations:**

1. Consider the system processes in which the components connected by this connector participates when determine which system L.O.S. goals apply to this connector. L.O.S. goals that apply to the system processes (SSAD 2.4) generally apply to the connectors that participate in the process.

2. Consider derived goals, i.e. a system–level throughput requirement may mean imply a maximum time to complete one of this connector’s processes (e.g. transporting a message).

3. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

**Common Pitfalls:**

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.

**Model Integration Rules:**

- Each connector L.O.S. Goals (OCD 4.4) should be described here.
- Each process listed here, should be described in the connector’s Processes (SSAD 3.1.6.1.4.1).
- Each mode listed here, should be described in the system’s Modes of Operation (SSAD 3.1.6.1.4.2).
- Each artifact or information classifier listed here, should be described in the architecture’s Analysis Classes (SSAD 3.2).

**3.1.6.1.6 Constraints**

Describe any constraints on the use and implementation of this connector that are not captured in other sections of the connector description. The following paragraphs describe some typical constraints.

- Rules that the connector must implement in support of rules that the system must satisfy;
- Constraints imposed on connector’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on connector’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the connector’s design and implementation, without getting into implementation details.
Guidelines for MBASE System and Software Architecture Description (SSAD)

Representation:

Create a list of system rules described in either informal text or a formal specification language (e.g. UML’s OCL). Clearly identify any system processes (SSAD 2.3.1), modes (SSAD 2.3.2), or artifact and information classifiers (SSAD 2.2) to which the rule is known to apply.

Create a table like that shown in Table 17 that shows how organization rules are supported by system rules and to which system elements the organization and system rules. If an organization rule is not supported by a system rule, put “N/A” (not applicable) in the System Rule column.

Table 17: Map of Connector Constraints to Sources and Elements

<table>
<thead>
<tr>
<th>Trace’s Back To</th>
<th>Connector Rule</th>
<th>Applies To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier of System Rule or Architecture Style, Pattern Or Framework</td>
<td>Connector Rule Identifier</td>
<td>List of connector’s processes, modes, or artifact or information classifiers</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Trade-offs:

See Trade-offs in section OCD 3.5(current organization rules).

Recommendations:

See Recommendations in section OCD 3.5(current organization rules).

1. To facilitate traceability, assign each rule a unique number (e.g. Connector-Rule-01 or CR-01).

Common Pitfalls:

- Describing a rule that is already captured in another view of the connector (e.g. modes of operation (SSAD 3.1.6.1.4.2)).

Model Integration Rules:

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each process described in a constraint, should be described in the connector’s Processes (SSAD 3.1.6.1.4.1).
- Each mode described in a constraint, should be described in the system’s Modes of Operation (SSAD 3.1.6.1.4.2).
- Each artifact or information classifier described in a constraint, should be described in the architecture’s Analysis Classes (SSAD 3.2).
- IOC: Each connector rule should be implemented by one or more elements described in either this connector’s Internal Architecture (SSAD 3.1.6.1.7), or its Implementation Design (SSAD 4).
3.1.6.1.7 Internal Architecture

Describe the architecture of this hardware connector classifier that is independent of the implementation technology. Describe the subconnectors of this connector, what they are expected to do, how they are connected, and what they communicate.

Representation:

Apply the same guidelines to describe the architecture of a connector as was used to describe for the system architecture in section Architecture Design & Analysis (SSAD 3), i.e. create subsections: “Structure”, “Analysis Classes”, “Behavior”, “L.O.S., Projected”, and “Architectural Styles, Patterns & Frameworks”.

Recommendations:

1. For a System of System, just include a reference to the project documents (e.g. OCD, SSAD) for the project that is responsible for creating the subsystem represented by this connector.

2. Only specify the internal architecture for a hardware connector when the connector will to be implemented by the project or when the internal architecture is needed to analyze the system. For a Small System or Composite System, the internal architectures of hardware connectors are often not important.

3. You need not specify the architecture of a connector that according to your architecture language or style is the lowest level that can be specified (i.e. “atomic”). A complete architecture specification would define an internal architecture for all but the atomic connectors. However, schedule or cost constraints may stop the architecture design process before uniformly reaching all atomic connectors. A project may also stop if it determines that there is little benefit from specifying all atomic connectors.

Common Pitfalls:

- Spending time describing the internal architecture for low risk connectors, including connectors that will not be implemented by this project. (This problem is more common for Simple Systems and small Composite Systems.) If the connector is low risk, consider deferring its details until Implementation Design (SSAD 4).

577 Guidelines:

While your system is probably a Composite System, it is probably simple enough, that you are unlikely to need this detailed a description for your hardware. (It is probably less risk to defer describing the hardware’s internal architecture until Implementation Design (SSAD 4) or to not describe the hardware’s internal architecture at all.)

If you decide this level of detail is needed, create a UML package with the stereotype <<systemModel>> the name “Architecture” in the package representing this component (i.e. the one with stereotype <<node>> and the name “Connector Classifier Name AD”).

3.1.7 Software Component Classifiers

For each software component classifier defined in the figure(s) shown in the Software Classifier Model (SSAD 3.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the software component classifier in the header.

Representation:

The description of software components classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.
Common Pitfalls:

For Simple Systems or Composite Systems, there is a risk of spending too much time describing the behavior (3.1.5.1.4) or the internal architecture (3.1.5.1.7) of a component in detailed.

Model Integration Rules:

- At LCO: Draft components are identified. Each component shall have a clear statement of purpose.
- At LCA: Components and interfaces of components that are high–risk shall be described in sufficient detail to support evaluation of correctness, and have a draft implementation design (SSAD 4.1).
- At IOC: All components and interfaces of components in the build shall be described in sufficient detail to support evaluation of correctness, and have a designed implementation (SSAD 4.1).

577 Guidelines:

Create a UML package, with the stereotype <<component>> and the name “Component Classifier Name AD”, in the package that holds the classifier that represents this software component classifier.

3.1.7.1 Component Classifier X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill–in the details for the following sections.

3.1.7.1.1 Purpose

Describe the purpose of this class of component.

3.1.7.1.2 Interface(s)

Describe the visible features of the component classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

Representation:

Describe the visible features of the component classifier using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Static–Structure Diagrams & Component Diagrams. A set of services is called an interface, which is represented as a classifier with the stereotype <<interface>> and the name of the set (e.g. “Security Services” or “File Management”). Note: an interface in UML is not allowed to have either attributes or parts.

Create a UML Static–Structure Diagram named “Interface Details” that defines the interfaces of the software component classifier, the operations they contained, and any inheritance relations among them. (Note: these interface classifiers may be used to define multiple components, connectors, and classes.)

Create a UML Component Diagram named “Interfaces” that shows the component classifiers for the component being described and its visible subcomponents; the interface classifiers (they will appear in icon form) define in the Static–Structure Diagram, and its visible attributes. Connect the component representing the component being described to each interface with an association. To represent a visible subcomponent to the component being described, either (a) connect the component representing the
Guidelines for MBASE System and Software Architecture Description (SSAD)

component being described using a composition relation to each component that represents a subcomponent; or (b) graphically nest each component classifier representing a subcomponent in the component classifier representing the component being described. Set the visibility of each subcomponent to public. (Operations in interfaces are always public.)

(UML does not allow attributes for the components.)

Note: the component’s Interface(s) (SSAD 3.1.7.1.2) and Parameters (SSAD 3.1.7.1.3) can be represented using one Component Diagram. (Name the diagram “Interfaces & Parameters”.)

For each feature or set of features, create a subsection numbered 3.1.7.1.2.x and the name of the software feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Model Integration Rules:

- Each interface feature described here, should be used in the component’s behavior description (SSAD 3.1.7.1.4).
- At LCA: a preliminary set of interfaces for the component shall be described.
- At IOC: the final set of interfaces for the component shall be described.

577 Guidelines:

Rose does support the current UML semantics for the Component Diagram, including some of what is described above. Create the Static–Structure Diagram named “Interfaces”.

- Add a classifier with the stereotype <<component>> for the component being described and for each of its visible subcomponents.
- Add a classifier with the stereotype <<interface>> and the name of the interface for each set of related operations. Define the operations they contained and show any inheritance relations among the interfaces.
- Connect the <<component>> classifier representing the component being described to each interface with a realization relation.
- Add any attributes of the software component to the <<component>> classifier.
- Add a classifier with the stereotype <<component>> for each of its visible subcomponents.
- Add a composition relation from the <<component>> classifier representing the component being described to each <<component>> classifier that represents a subcomponent. (If the <<component>> classifier representing the component being described does not have any attributes, then graphically nest each <<component>> classifier that represents a subcomponent inside the component representing the component being described can be an alternate representation for composition; but many tools do not recognize the relation.)
- Set the visibility of each subcomponent and each attribute to public. (Operations in interfaces are always public.)

Note: the component’s Interface(s) (SSAD 3.1.7.1.2) and Parameters (SSAD 3.1.7.1.3) can be represented using one Static–Structure Diagram. (Name the diagram “Interfaces & Parameters”.)
3.1.7.1.2.1 Feature or Feature Set X

Create a section at this level for each feature or feature set described above. The header of this section should be the name of the feature or feature set and its unique designator, if you have assigned an identifier and it is different from the name. The contents of this section should include a description of the feature’s purpose and the following:

- If the feature is a attribute, include its purpose, its classifier and any default value;
- If the feature is a component that is a part, include a reference to its description in the Internal Architecture (SSAD 3.1.7.1.7) section;
- If the feature is an operation, include a description of its purpose, its parameters and result (called a signature), and any pre– and post–condition;
- If describing a interface, describe the each member of the interface.

3.1.7.1.3 Parameters

Describe any parameters of the software component classifier. The parameters need to be set when an instance of the software component classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other components.

Representation:

Describe the parameters of the component classifier using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Component Diagrams. Create a UML Component Diagram that shows the component classifier for the component being described. Add a dashed rectangular box to upper–right hand corner of the component classifier. List the parameters in the box.

Note: the component’s Interface(s) (SSAD 3.1.7.1.2) and Parameters (SSAD 3.1.7.1.3) can be represented using one Component Diagram.

Create a table like Table 13 (page 6) that lists each parameter.

Model Integration Rules:

- At LCA: a preliminary set of parameters for the component, if any exist, shall be described.
- At IOC: the final set of parameters for the component shall be described. (If none, say so.)

577 Guidelines:

Rose does not fully support the current UML semantics for the Component Diagram, including what is described above. Create a Static–Structure Diagram instead of the UML Component Diagram described above. In the Static–Structure Diagram, create a classifier with the stereotype <<component>> for the component being described. Add a dashed rectangular box to upper–right hand corner of the classifier. List the parameters in the box.

Note: the component’s Interface(s) (SSAD 3.1.7.1.2) and Parameters (SSAD 3.1.7.1.3) can be represented using one Component Diagram.
3.1.7.1.4 Behavior

Describe behavior of the instances of this component classifier.

Model Integration Rules:

- At LCA: a draft behavior for the component shall be described if the component is high-risk.
- At IOC: the behavior for the component shall be described if the component is included in the build.

UML Guidelines:

Describe the processes of this component classifier, including how it works with the other components and the system’s actors, and how it uses the artifacts and information; and if significant, how the component’s behavior depends on its mode.

The following subsection work for UML–based architecture descriptions and may work for other ADL’s.

3.1.7.1.4.1 Processes

Describe the processes of this component classifier. For each process, identify which other components and actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by this component and each actor during the process.

Representation:

As in 4.3, create a Use–Case Model that describes the component’s processes. Represent the model as a package with the stereotype <<use–case model>> with the package name “Component Classifier Name Processes”.

- Create one or more Use–Case Diagrams
- For each process, create a Use–Case Description, and an Activity Diagram.

For each process, create a subsection numbered 3.1.7.1.4.1.x (see example section below) with the name of the process in the header. Each subsection should provide the information shown in the Figure 4 and an Activity Diagram.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

See Recommendations in 4.3.

1. Use the “just do it” approach to eliminate the pressure to get go right set of processes on the first pass (like writing a rough draft for a term paper). Each process may require several iterations to get right. “Go with what you know”, then revise the set of processes and adjust descriptions, as needed.

2. Start with the component’s actions or requested operation described in the architecture behavior description (SSAD 3.3).

3. To facilitate traceability, assign each process a unique number (e.g. Process-01 or UC-01).

4. For LCA
The following fields of Use–Case Description should be filled out for all use–cases: Name, Purpose, Actors, Importance, Requirements, Development Status, Pre–conditions, Post–conditions, and Includes.

Typical, Alternate, and Exceptional Courses of Actions should be described for high–risk, architecturally significant, or particularly complex use–case. Fill–in the Includes & Extension Points fields if the courses of actions specify the inclusion of other use–cases or extension point, respectively.

5. For IOC, use–cases descriptions should be completely filled out for all use–cases.

Common Pitfalls:

See Common Pitfalls in 4.3.

Model Integration Rules:

- Each operation defined in the component’s interface (SSAD 3.1.7.1.2) must either be represented by a use–case or represented as an action in a use–case’s Activity Diagram.

- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).

- Each component in a process description should be defined in the architecture of the system (SSAD 3.1).

- Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information for the system’s architecture (SSAD 3.2).

3.1.7.1.4.1.1 Process X

Create a section at this level for each process of the component. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.

3.1.7.1.4.2 Modes of Operation

Describe the modal behavior of the component instances, i.e. how the behavior of the component depends on the mode it is in. Describe the high–level states of the component, the events that cause the component to change modes, and how the processing is different in each mode.

Representation:

As in SSAD 2.3.2, create a State Model. Represent the model as a package with the stereotype <<state model>> with the package name “Modes of Component Classifier Name”. Create one or more Statecharts that show the modes of component and the events that cause mode changes.

For each mode, create a subsection numbered 3.1.7.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list component’s processes (SSAD 3.1.7.1.4.1) are available in that mode (see Table 14 on page 6). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode–specific behavior).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right–click on the <<component>> classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity
Model” that is attached to the <<component>> classifier, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

**Recommendations:**

See Recommendations in SSAD 2.3.2

**Common Pitfalls:**

See Common Pitfalls in SSAD 2.3.2

**Model Integration Rules:**

- Each component process (SSAD 3.1.7.1.4.1) should be listed as available in one or more modes.
- Each process listed as available in one or more modes should appear in described in SSAD 3.1.7.1.4.1.

### 3.1.7.1.4.2.1 Mode X

Create a section at this level for each mode of the component. The header of this section should be the name of the mode and its unique designator, if you have assigned an identifier and it is different from the name. Describe the mode (e.g. for the airplane’s taxi mode, “when in this mode, the airplane is moving on the ground, usually to or from a gate”).

List the component’s processes (SSAD 3.1.7.1.4.1) that are available in this mode (see Table 14). For each process, describe any effect of the mode.

### 3.1.7.1.5 L.O.S. Goals

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component classifier. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a component classifier defines the goals for all instances of this classifier, and the goals that an implementation of this component classifier must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the system’s goals (i.e. the system goals will be meet its goals if all connectors and components achieve their goals).

**Representation:**

For each system goal that applies to this component class fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its on value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

1. Consider the system processes in which instances of this component classifier participates, when determining which system L.O.S. goals apply to this component classifier. L.O.S. goals that apply to the system processes (SSAD 2.4) generally apply to the components that participate in the process.

2. Consider derived goals, i.e. a system–level throughput requirement may mean imply a maximum time to complete one of this component classifier’s processes.
3. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

**Common Pitfalls:**

- Not satisfying the M.R.S. criteria.

**Model Integration Rules:**

- Each component L.O.S. Goals (OCD 4.4) should be described here.
- Each process listed here, should be described in the component’s Processes (SSAD 3.1.7.1.4.1).
- Each mode listed here, should be described in the system’s Modes of Operation (SSAD 3.1.7.1.4.2).
- Each artifact or information classifier listed here, should be described in the architecture’s Analysis Classes (SSAD 3.2).

### 3.1.7.1.6 Constraints

Describe any constraints on the use and implementation of this component that are not captured in other sections of the component description. The following paragraphs describe some typical constraints.

- Rules that the component must implement in support of rules that the system must satisfy;
- Constraints imposed on component’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on component’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the component’s design and implementation, without getting into implementation details.

**Representation:**

Create a list of system rules described in either informal text or a formal specification language (e.g. UML’s OCL). Clearly identify any system processes (SSAD 2.3.1), modes (SSAD 2.3.2), or artifact and information classifiers (SSAD 2.2) to which the rule is known to apply.

Create a table like that shown in Table 15 that shows how organization rules are supported by system rules and to which system elements the organization and system rules. If an organization rule is not supported by a system rule, put “N/A” (not applicable) in the System Rule column.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Trade-offs:**

See Trade-offs in section OCD 3.5(current organization rules).

**Recommendations:**

See Recommendations in section OCD 3.5(current organization rules).

1. To facilitate traceability, assign each rule a unique number (e.g. Component-Rule-01 or CR-01).
Common Pitfalls:

- Describing a rule that is already captured in another view of the component (e.g. modes of operation (SSAD 3.1.7.1.4.2)).

Model Integration Rules:

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each process described in a constraint, should be described in the component classifier’s Processes (SSAD 3.1.7.1.4.1).
- Each mode described in a constraint, should be described in the component classifier’s Modes of Operation (SSAD 3.1.7.1.4.2).
- Each artifact or information classifier described in a constraint, should be described in the architecture’s Analysis Classes (SSAD 3.2).
- IOC: Each component rule should be implemented by one or more elements described in either this component’s Internal Architecture (SSAD 3.1.7.1.7), or its Implementation Design (SSAD 4).

3.1.7.1.7 Internal Architecture

Describe the architecture of this software component classifier that is independent of the implementation technology. Describe the subcomponents of this component, what they are expected to do, how they are connected, and what they communicate.

Representation:

Apply the same guidelines to describe the architecture of a component as was used to describe for the system architecture in section Architecture Design & Analysis (SSAD 3), i.e. create subsections: “Structure”, “Analysis Classes”, “Behavior”, “L.O.S., Projected”, and “Architectural Styles, Patterns & Frameworks”.

Recommendations:

1. For a System of System, just include a reference to the project documents (e.g. OCD, SSAD) for the project that is responsible for creating the subsystem represented by this component.

2. Only specify the internal architecture for a hardware component when the component will to be implemented by the project or when the internal architecture is needed to analyze the system. For a Small System or Composite System, the internal architectures of hardware components are often not important.

3. You need not specify the architecture of a component which according to your architecture language or style is the lowest level that can be specified (i.e. “atomic”). In UML, an atomic component is the lowest–level modular, deployable, and replaceable part of system (e.g. the representation of an executable, a link–library, a Java Been).

A complete architecture specification would define an internal architecture for all but the atomic components. However, schedule or cost constraints may stop the architecture design process before uniformly reaching all atomic components. A project may also stop if it determines that there is little benefit from specifying all atomic components.
Common Pitfalls:

- Spending time describing the internal architecture for low risk components, including components that will not be implemented by this project. (This problem is more common for Simple Systems and small Composite Systems.) If the component is low risk, consider deferring its details until Implementation Design (SSAD 4).

Model Integration Rules:

- At LCA: an internal architecture for each non–atomic component shall be described if the component is high–risk or complex.
- At IOC: an internal architecture for each non–atomic component shall be described if the component is included in the build.

577 Guidelines:

While your system is probably a Composite System, it is probably simple enough, that you are unlikely to need this detailed of a description for your software.

If you decide the component’s internal architecture needs to be specified, create a UML package with the stereotype <<Structure Model>> the name “Architecture” in the package representing this component (i.e. the one with stereotype <<component>> and the name “Component Classifier Name AD”) to hold the internal architecture description.

3.1.8 Software Connector Classifiers

For each software connector classifier defined in the figure(s) shown in the Software Classifier Model (SSAD 3.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the software connector classifier in the header.

Note:

The description of software connector classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

Software Connectors are a relatively new concept in software architectures. Few ADL’s represent software connectors in any details.

The subsections of this section are intended to support a wide range of systems, architecture languages, and architecture styles.

Model Integration Rules:

- At LCO: Draft connectors are identified. Each connector shall have a clear statement of purpose.
- At LCA: Connectors and interfaces of connectors that are high–risk shall be described in sufficient detail to support evaluation of correctness, and have a draft implementation design (SSAD 4.1).
- At IOC: All connectors and interfaces of connectors in the build shall be described in sufficient detail to support evaluation of correctness, and have a designed implementation (SSAD 4.1).

UML Guidelines:

The current definition of UML [OMG 2001] does not define software connectors. If you are using the UML standard without extension, delete this section.
3.1.8.1 Connector Classifier X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

3.1.8.1.1 Purpose

Describe the purpose of this class of connector.

3.1.8.1.2 Interface(s)

Describe the visible features of the connector classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

Representation:

Describe the visible features of the connector classifier using an Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

For each feature or set of features, create a subsection numbered 3.1.8.1.2.x and the name of the software feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

- Each interface feature described here, should be used in the connector’s behavior description (SSAD 3.1.7.1.4).
- At LCA: a preliminary set of interfaces for the connector shall be described.
- At IOC: the final set of interfaces for the connector shall be described.

3.1.8.1.2.1 Feature or Feature Set X

Create a section at this level for each feature or feature set described above. The header of this section should be the name of the feature or feature set and its unique designator, if you have assigned an identifier and it is different from the name. The contents of this section should include a description of the feature’s purpose and the following:

- If the feature is a attribute, include its classifier and any default value;
- If the feature is a component that is a part, include a reference to its description in the Internal Architecture (SSAD 3.1.7.1.7) section;
- If the feature is an operation, include a description of its parameters and result (called a signature), and any pre- and post-condition;
- If describing a feature set, describe the each member of the set as defined in the previous three bullets.
3.1.8.1.3 Parameters

Describe any parameters of the software connector classifier. The parameters need to be set when an instance of the connector classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other connectors.

**Representation:**

Describe the parameters of the connector classifier using either An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

Create a table like Table 13 (on page 6) that lists each parameter.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Model Integration Rules:**

- At LCA: a preliminary set of parameters for the connector, if any, shall be described.
- At IOC: the final set of parameters for the connector, if any, shall be described.

3.1.8.1.4 Behavior

Describe behavior of the instances of this connector classifier.

**Representation:**

Describe the behavior of the connector classifier using an Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

3.1.8.1.5 L.O.S. Goals

Describe how the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be **measurable, relevant, and specific** (M.R.S.).

The L.O.S. Goals for a connector classifier defines the goals for all instances of this classifier, and the goals that an implementation of this connector classifier must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the system’s goals (i.e. the system goals will be meet its goals if all connectors and components achieve their goals).

**Representation:**

For each system goal that applies to this connector fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its on value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

**Recommendations:**

1. Consider the system processes in which the components connected by this connector participates when determine which system L.O.S. goals apply to this connector. L.O.S. goals that apply to the system processes (SSAD 2.4) generally apply to the connectors that participate in the process.
2. Consider derived goals, i.e. a system-level throughput requirement may mean imply a maximum time to complete one of this connector’s processes (e.g. transporting a message).

3. Include both desired and acceptable levels, when possible. Since some L.O.S. goals conflict (e.g., performance and fault-tolerance), specifying desired and acceptable levels allows for more flexibility during design and implementation.

Common Pitfalls:

- Not satisfying the M.R.S. criteria.

Model Integration Rules:

- Each connector L.O.S. Goals (OCD 4.4) should be described here.
- Each artifact or information classifier listed here, should be described in the architecture’s Analysis Classes (SSAD 3.2).

3.1.8.1.6 Constraints

Describe any constraints on the use and implementation of this connector that are not captured in other sections of the connector description. The following paragraphs describe some typical constraints.

- Rules that the connector must implement in support of rules that the system must satisfy;
- Constraints imposed on connector’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on connector’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the connector’s design and implementation, without getting into implementation details.

Representation:

Create a list of system rules described in either informal text or a formal specification language (e.g. UML’s OCL). Clearly identify any system processes (SSAD 2.3.1), modes (SSAD 2.3.2), or artifact and information classifiers (SSAD 2.2) to which the rule is known to apply.

Create a table like that shown in Table 17 that shows how organization rules are supported by system rules and to which system elements the organization and system rules. If an organization rule is not supported by a system rule, put “N/A” (not applicable) in the System Rule column.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Trade-offs:

See Trade-offs in section OCD 3.5(current organization rules).

Recommendations:

See Recommendations in section OCD 3.5(current organization rules).

1. To facilitate traceability, assign each rule a unique number (e.g. Connector-Rule-01 or CR-01).
Common Pitfalls:

- Describing a rule that is already captured in another view of the connector (e.g. behavior (SSAD 3.1.8.1.4)).

Model Integration Rules:

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each artifact or information classifier described in a constraint, should be described in the architecture’s Analysis Classes (SSAD 3.2).
- IOC: Each connector rule should be implemented by one or more elements described in either this connector’s Internal Architecture (SSAD 3.1.8.1.7), or its Implementation Design (SSAD 4).

### 3.1.8.1.7 Internal Architecture

Describe the architecture of this software connector classifier that is independent of the implementation technology. Describe the subcomponents of this connector, what they are expected to do, how they are connected, and what they communicate.

**Representation:**

Apply the same guidelines to describe the architecture of a connector as was used to describe for the system architecture in section Architecture Design & Analysis (SSAD 3), i.e. create subsections: “Structure”, “Analysis Classes”, “Behavior”, “L.O.S., Projected”, and “Architectural Styles, Patterns & Frameworks”.

**Recommendations:**

1. For a System of System, just include a reference to the project documents (e.g. OCD, SSAD) for the project that is responsible for creating this connector.

2. Only specify the internal architecture for a hardware connector when the connector will be implemented by the project or when the internal architecture is needed to analyze the system. For a Small System or Composite System, the internal architectures of hardware connectors are often not important.

3. You need not specify the architecture of a connector which according to your architecture language or style is the lowest level that can be specified (i.e. “atomic”). A complete architecture specification would define an internal architecture for all but the atomic connectors. However, schedule or cost constraints may stop the architecture design process before uniformly reaching all atomic connectors. A project may also stop if it determines that there is little benefit from specifying all atomic connectors.

**Common Pitfalls:**

- Spending time describing the internal architecture for low risk connectors, including connectors that will not be implemented by this project. (This problem is more common for Simple Systems and small Composite Systems.) If the connector is low risk, consider deferring its details until Implementation Design (SSAD 4).

**Model Integration Rules:**

- At LCA; an internal architecture for each non–atomic connector shall be described if the component is high–risk or complex.
• At IOC: an internal architecture for each non-atomic connector shall be described if the component is included in the build.

### 3.1.9 Hardware Components

For each hardware component defined in the figure(s) shown in the Hardware Classifier Model (SSAD 3.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware component classifier in the header.

**Note:**

The description of hardware components classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

#### 3.1.9.1 Component \( X \)

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

#### 3.1.9.1.1 Purpose

Describe the purpose of this component.

#### 3.1.9.1.2 Classifier

Identify the classifier of this component.

#### 3.1.9.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be **measurable, relevant, and specific** (M.R.S.).

The L.O.S. Goals for a component classifier (SSAD 3.1.5.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this component must satisfy the values specified either by the classifier or this instance.

**Representation:**

Create a table like Table 18, where a value is specified for each goal that applies to this component’s classifier (SSAD 3.1.5.1.5), where the classifier described as “value is specific to the component instance”.

<table>
<thead>
<tr>
<th>Goal id &amp; description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal 1</td>
<td>Value 1</td>
</tr>
<tr>
<td>Goal 2</td>
<td>Value 2</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Model Integration Rules:**

- Each goal described here, should be described in the L.O.S. Goals of the component’s classifier (SSAD 3.1.5.1.5).
• LCA: If this component is high-risk, then each goal described in the L.O.S. Goals of the component’s classifier (SSAD 3.1.5.1.5) described as “value is specific to the component instance”, shall have a value.

• IOC: Each goal described in the L.O.S. Goals of the component’s classifier (SSAD 3.1.5.1.5) described as “value is specific to the component instance”, shall have a value.

3.1.10 Hardware Connectors

For each hardware connector defined in the figure(s) shown in the Hardware Classifier Model (SSAD 3.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware connector classifier in the header.

Note:

The description of hardware connectors classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

3.1.10.1 Connector X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

3.1.10.1.1 Purpose

Describe the purpose of this connector.

3.1.10.1.2 Classifier

Identify the classifier of this connector.

3.1.10.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a connector classifier (SSAD 3.1.6.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this connector must satisfy the values specified either by the classifier or this instance.

Representation:

Create a table like Table 18, where a value is specified for each goal that applies to this connector’s classifier (SSAD 3.1.6.1.5), where the classifier described as “value is specific to the connector instance”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

• Each goal described here, should be described in the L.O.S. Goals of the connector’s classifier (SSAD 3.1.6.1.5).

• LCA: If this connector is high-risk, then each goal described in the L.O.S. Goals of the connector’s classifier (SSAD 3.1.6.1.5) described as “value is specific to the connector instance”, shall have a value.
• IOC: Each goal described in the L.O.S. Goals of the connector’s classifier (SSAD 3.1.6.1.5) described as “value is specific to the connector instance”, shall have a value.

3.1.11 Software Components

For each hardware component defined in the figure(s) shown in the Software Classifier Model (SSAD 3.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware component classifier in the header.

Note:

The description of hardware components classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

3.1.11.1 Component X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

3.1.11.1.1 Purpose

Describe the purpose of this component.

3.1.11.1.2 Classifier

Identify the classifier of this component.

3.1.11.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a component classifier (SSAD 3.1.7.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this component must satisfy the values specified either by the classifier or this instance.

Representation:

Create a table like Table 18, where a value is specified for each goal that applies to this component’s classifier (SSAD 3.1.7.1.5), where the classifier described as “value is specific to the component instance”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

• Each goal described here, should be described in the L.O.S. Goals of the component’s classifier (SSAD 3.1.7.1.5).

• LCA: If this component is high-risk, then each goal described in the L.O.S. Goals of the component’s classifier (SSAD 3.1.7.1.5) described as “value is specific to the component instance”, shall have a value.

• IOC: Each goal described in the L.O.S. Goals of the component’s classifier (SSAD 3.1.7.1.5) described as “value is specific to the component instance”, shall have a value.
3.1.12 Software Connectors

For each hardware connector defined in the figure(s) shown in the Software Classifier Model (SSAD 3.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware connector classifier in the header.

**Note:**

The description of hardware connectors classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

### 3.1.12.1 Connector X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

#### 3.1.12.1.1 Purpose

Describe the purpose of this connector.

#### 3.1.12.1.2 Classifier

Identify the classifier of this connector.

#### 3.1.12.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be *measurable, relevant, and specific* (M.R.S.).

The L.O.S. Goals for a connector classifier (SSAD 3.1.8.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this connector must satisfy the values specified either by the classifier or this instance.

**Representation:**

Create a table like Table 18, where a value is specified for each goal that applies to this connector’s classifier (SSAD 3.1.8.1.5), where the classifier described as “value is specific to the connector instance”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Model Integration Rules:**

- Each goal described here, should be described in the L.O.S. Goals of the connector’s classifier (SSAD 3.1.8.1.5).
- LCA: If this connector is high-risk, then each goal described in the L.O.S. Goals of the connector’s classifier (SSAD 3.1.8.1.5) described as “value is specific to the connector instance”, shall have a value.
- IOC: Each goal described in the L.O.S. Goals of the connector’s classifier (SSAD 3.1.8.1.5) described as “value is specific to the connector instance”, shall have a value.


3.2 Analysis Classes

Create a model of the information classes\(^\text{11}\) that are needed to support the architectural structure (SSAD 3.1) implement the system behavior (SSAD 3.3). This class model may include classes representing following:

- artifacts and information required by the system (SSAD 2.2);
- forms defined in the current prototype (OCD 5);
- information needed for communication components in the architectural structure (SSAD 3.1);
- control behavior specific to one or a few processes performed by architectural units (i.e. “recipes” that define how to do something).

**Representation**

Create a class model using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. UML’s Class Model. Represent the model as a package with the stereotype <<Class Model>> and the name “Analysis Classes”. In the model Analysis Class packages, create one or more Static–Structure Diagrams to show the information classes that are needed to support the architectural structure implement the system behavior, and their relations.

Represent each artifact or information classifier that is stored, or used for internal communication as a classifier with the stereotype <<entity>>. Represent each form or other information that is used to communicate with an actor as a classifier with the stereotype <<boundary>>. (Some artifacts and information classifier may have both <<entity>> and <<boundary>> representations.)

If one class is a specialization of class (e.g. Tax Report and Report), then show a generalization relation from the classifier representing the specialized class to the classifier representing the generalized class.

If two classes are directly related other than by specialization, then an association is drawn to connect the two classifiers. The association should be labeled with the name of the relation. Each end of the association may be labeled with the role that class plays in the relation and the number of instances of the class that can be related to one incidents of the class at the other end of the association.

If two classes are not directly related; but one needs the other (e.g. as the class of a parameter), then a dependency relation is drawn from dependant class (“client”) to other class (“supplier”). The dependency relation may have a name (uncommon) and a stereotype.

If there are a large number of classes, then it is often useful to organize the classes and diagrams using packages.

Each Static–Structure Diagram should be accompanied by a brief description of its purpose.

For each class represented, create a subsection numbered 3.2x (see example section below) and the name of the artifact in the header. The text of the subsection should give a brief description of artifact or class of information.

\(^{11}\) Some ADL’s use the term type rather than class.
Guidelines for MBASE System and Software Architecture Description (SSAD)

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

Recommendations:

1. Give each classifier a name that expresses the artifact or information that it represents.
   - A good name is usually a noun or short noun phrase. Avoid phrases that imply state of an object, e.g. "completed document".
   - Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   - Clear, self-explanatory names may require several words.

2. To facilitate traceability, assign each class a unique number (e.g. Entity-01 or Boundary-01)

3. Focus on classes and their details that are needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture. Provided additional class and details during implementation design (SSAD 4).

4. Create additional Static–Structure Diagrams that show how each artifact and information classifier defined during system analysis (SSAD 2.2) is implemented by one or more analysis classes and how the information that the system needs to maintain about an actor is represented by one or more analysis. Represent each system artifact and information classifier as described in system analysis (SSAD 2.2), and each of the architecture’s analysis class as described under Representation in this section. For each analysis class that implements a system artifact or information classifier, show a dependency relation with the stereotype <<trace>> from the analysis class to the system artifact or information classifier that the analysis class implements. For each analysis class that represents information about an actor, show a dependency relation with the stereotype <<trace>> from the analysis class to the actor whose information the analysis class represents.

5. Suppress the display of all (or all but the most important) attributes and operations on the Static–Structure Diagrams. Showing attributes or operations of the classes on the Static–Structure Diagram can results in diagrams that are too clutter and so difficult to understand.

Common Pitfalls:

- Including components (SSAD 3.1).
- At LCA:
  - Not representing all artifact and information classifiers define for the system (SSAD 2.2).
  - Including details about a class that are not needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture. (They should be deferred to implementation design (SSAD 4).)
- At IOC:

---

12 UML says that the trace relation is often bidirectional (i.e. a reader may trace from the source of the dependency to the target or visa versa) so the directionality of the dependency relation can be ignored.

13 UML allows for the use of refinement or realization relations instead of the trace relation.
Guidelines for MBASE

- Including classes that are not needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture.

- Including classes that is not represented (called realized) in implementation design (SSAD 4).

Model Integration Rules:

- IOC:
  - Each artifact and information kind artifact and information classifiers define for the system (SSAD 2.2) should be represented.
  - Each class needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture should be represented.

- LCA:
  - Each artifact and information classifiers define for the system (SSAD 2.2) shall be represented here.
  - Each class needed for specifying the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture shall be represented.

- IOC:
  - At least one instance of each class shall be used to describe either the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture.
  - Every detail about a class shall be used to describe either the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture.
  - Each class shall be realized in implementation design (SSAD 4).

577 Guidelines:

Create an Analysis Class Model using a UML Class Model as described above.

3.2.1 Analysis Class X

Create a section at this level for each artifact in the figure(s) shown in SSAD 3.2. The header of this section should be the name of the class and its unique designator, if you have assigned an identifier and it is different from the name.

Provide a brief description of its role, purpose, and responsibilities. List any attributes and operations that are used to describe either the structure (SSAD 3.1) or behavior (SSAD 3.3) of the architecture. List any system capabilities that use the artifact.

3.3 Behavior

Describe how the components work with each other and with the actors to implement the required behavior of the system (SSAD 2.3).

Describe how each system process (SSAD 2.3) is implemented by the components described in the Deployment Model (SSAD 3.1.4). For each process, identify which components participate in the system process, and which instances of the analysis classes (SSAD 3.2) are inspected, manipulated, or produced in the process; and describe the interactions among the components and the analysis classes.
Guidelines for MBASE System and Software Architecture Description (SSAD)

**Representation:**

Create a behavior model using either:

a. An Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

b. Create a Use–Case Model that describes the system’s processes, the actors that participate in each process, and the relations among the processes and the outside actors. Represent the model as a package with the stereotype <<use–case model>> with the package name “Process Implementations”.

   - Create one or more Use–Case Diagrams that show the system processes use–case, the process implementations that implement them, and a realization relation from each process implementation to the process that it implements. Represent each process as a basic use–case, and each implementation as a use–case with the stereotype <<use–case realization>>. (Some processes may have multiple implementations. For example, a Login process may have “Basic” implementation and a “Secure” implementation)

   - For each process implementation, create a Use–Case Realization Description, and one or more Interaction Diagram.

**Figure 5: Use–Case Realization Description**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Unique identifier for traceability (e.g. UCR-xx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use–case Realization Name</td>
<td>Name of process implementation</td>
</tr>
<tr>
<td>Use–case Realized</td>
<td>Name &amp; identifier of processes</td>
</tr>
<tr>
<td>Purpose</td>
<td>Brief description of purpose (include how it is different from other implementations)</td>
</tr>
<tr>
<td>Requirements</td>
<td>List of requirements that this use–case implementation satisfies</td>
</tr>
<tr>
<td>Risks</td>
<td>List of risks for this use–case realization</td>
</tr>
<tr>
<td>Development Status</td>
<td>Draft LCO</td>
</tr>
<tr>
<td>User Interface</td>
<td>Pictures and/or descriptions of user interface, if applicable, that is needed to describe the behavior (e.g. reference or URL to prototype screen).</td>
</tr>
<tr>
<td>Pre–conditions</td>
<td>Description of state of system and participants before use–case performed. (informal text or OCL, or both)</td>
</tr>
<tr>
<td>Post–conditions</td>
<td>Description of state of system and participants after use–case performed. (informal text or OCL, or both)</td>
</tr>
</tbody>
</table>

For each process implementation, create a subsection numbered 3.3.x (see example section below) with the name of the process implementation in the header. Each subsection should provide the information shown in the Figure 5 and one or more Interaction Diagrams.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

1. Create at least one implementation for each process.
2. Give each process implementation a name that expresses the behavior that it represents.

- A good name is usually a verb, or verb phrase.
- Each name must be unique relative to the containing package. (Two classifiers in different packages can have the same name.)
- Avoid names that sound alike or are spelled alike, as well as synonyms.
- Clear, self-explanatory names may require several words.

3. To facilitate traceability, assign each process a unique number (e.g. Process-Implementation-01 or UCR-01).

4. Use the “just do it” approach to eliminate the pressure to get go right set of processes on the first pass (like writing a rough draft for a term paper). Each implementation may require several iterations to get right. “Go with what you know”, then revise the set of implementation and adjust descriptions, as needed.

5. Start an Interaction Diagram by creating one or more instances of each actor that participates in the use-case realized. List the actions specified in the courses of action for the use-case realized along the left margin of the diagram. For each action,

- Identify component(s) of the system that should participate and instance(s) of analysis classes that are needed. (The activity diagram of the use-case should be helpful in identifying the analysis classes are involved.)
- Identify process(s) that each component should do. (Each process should be described in the behavior of the component’s class (SSAD 3.1.7.1.4.1).)
- Identify operations(s) on the instances of analysis classes need to be performed. (Each operation should be described in the analysis class’ description (SSAD 3.2.x)).
- Show message(s) that need to be exchanged.

6. Each process implementation should be described in enough detail to be testable.

**Common Pitfalls:**

- Simply creating one process for each capability (some processes will require more than one implementation).
- Describing a pre-condition then testing for that condition early in the Course of Action.
- Describing a post-condition than testing for that condition at the end of the Course of Action.
- Describing a condition has a precondition or post-condition when the condition must be tested in the Course of Action.

**Model Integration Rules:**

1. For LCO,

- Each system process (SSAD 2.3.1) that is high-risk, architecturally-significant, or particularly complex should have at least one implementation.
• A draft Use–Case Realization Description should be filled out.

• A draft Implementation Diagram should be created.
  • Each actor instance in the diagram should be an instance of an actor associated with the system process (SSAD 2.3.1).
  • Each non-actor instance in the diagram should be an instance of either a software component, or interface in the architecture (SSAD 3.1.11), or an analysis class (SSAD 3.2).
  • Each message in the diagram should be the name of a process, an operation, or an event defined for either a component or interface in the architecture (SSAD 3.1) or an analysis class (SSAD 3.2).
  • Each action described in courses of action for the use–case realized by one or more messages.

2. For LCA,

• Each system process (SSAD 2.3.1) that is high–risk, architecturally–significant, or particularly complex should have at least one implementation.

• Its Use–Case Realization Description shall be filled out.

• An Implementation Diagram shall be created.
  • Each actor instance in the diagram shall be an instance of an actor associated with the system process (SSAD 2.3.1).
  • Each non-actor instance in the diagram shall be an instance of either a software component or interface in the architecture (SSAD 3.1.11), or an analysis class (SSAD 3.2).
  • Each message in the diagram shall be the name of a process, an operation, or an event defined for either a component or interface in the architecture (SSAD 3.1) or an analysis class (SSAD 3.2).
  • Each action described in courses of action for the use–case realized by one or more messages.

3. For IOC, each system process (SSAD 2.3.1) shall have at least one implementation described to the same level of detail as a high–risk, architecturally–significant, or particularly complex implementation at LCO.

3.3.1.1 Process Realization X

Create a section at this level for each process implementation. The header of this section should be the name of the process implementation and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 5.

3.4 L.O.S., Projected

Describe the elements of the architecture (components, connectors, analysis classes, etc.) to which each system L.O.S. applies and the projected value of each system L.O.S.

The goal is to evaluate the L.O.S. that the architecture will provided (assuming each component, connector, and analysis class achieves its goals). This activity provides technical details supporting the FRD.
Guidelines for MBASE System and Software Architecture Description (SSAD)

Representation:

Create a table like Table 19, describing the following information for each system L.O.S. goal (SSAD 2.4).

- The architectural elements to which the L.O.S. goal applies.
- How the L.O.S. requirement applies, when the L.O.S. applies to more than one element.
- The current projected value.
- How the value was determined.

<table>
<thead>
<tr>
<th>L.O.S. Goal</th>
<th>Applies To</th>
<th>How</th>
<th>Projected Value</th>
<th>Evaluation Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name &amp; Identifier</td>
<td>List of architecture elements</td>
<td>If L.O.S. goal applies to more than one element, describe how it applies.</td>
<td>Determined value</td>
<td>Technique used to determine value (e.g., estimation, prototype, calculation)</td>
</tr>
</tbody>
</table>

When a system L.O.S. goal applies to multiple elements, the L.O.S. requirement generally applies to the elements in one of three ways.

- The requirement applies the same way to the elements (i.e. “equally”, “uniformly”).
- The requirement applies in equal portions to the elements. The requirement value is the average of the value for each element.
- The requirement applies in unequal portions to the elements. The requirement value is typically the weighted average of the value for each element.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. When possible, determine value by prototype or by a calculation based on sound theory (e.g. Rate Monotonic Analysis for scheduling).

2. L.O.S. goals specified here should be more specific than the L.O.S. Requirements (SSRD 5) or the system–level L.O.S. Goals (OCD 4.4).

Common Pitfalls:

- Not allocating requirements to specific architecture elements.
- Not repairing an architecture whose projected value for a L.O.S. does not satisfy the system goal.

Model Integration Rules:

- Each system L.O.S. goal (SSAD 2.4) should be described here.
- Each L.O.S. listed here, should be described in the system’s L.O.S. Goals (SSAD 2.4).
3.5 Architectural Styles, Patterns & Frameworks

Describe any architectural styles (e.g. the Prism style [http://sunset.usc.edu/~softarch/Prism/]), patterns (e.g., pipe-and-filter and client–server), or frameworks used to describe the system architecture.

**Representation:**

Create a table like Table 19, describing for style, pattern, and framework.

<table>
<thead>
<tr>
<th>Name &amp; Identifier</th>
<th>Description</th>
<th>Benefits, Costs, &amp; Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name &amp; Identifier</td>
<td>Briefly describe the style, pattern, or framework; provide a reference to its detailed description.</td>
<td>Summarize key benefits, costs, and limitations of using the style, pattern, or framework in this architecture.</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

1. Focus on benefits and costs that are relevant to the system’s L.O.S. Goals (SSAD 2.4).

**Common Pitfalls:**

- Specifying implementation-level styles, patterns, or frameworks. For example, the patterns described in the book *Design Patterns* [Gamma 19988] are general implementation-design patterns. These patterns should be described in SSAD 4.4 during implementation design.

**Model Integration Rules:**

- Each style, pattern, or framework should be used to define the architecture’s structure (SSAD 3.1) or its analysis classes (SSAD 3.2).

- Each benefit and cost described here, should be related to the system’s L.O.S. Goals (SSAD 2.4), and should be factored in to the projected L.O.S. for the architecture (SSAD 3.4).

- The details of each limitation should be captured in the architecture’s structure (SSAD 3.1), its analysis classes (SSAD 3.2).

4. Implementation Design

Design a technology-specific implementation for the system by refining the general architecture defined during Architecture Design & Analysis (SSAD 3).

- Describe the development technologies that are to be used in the implementation, including: hardware types (e.g. SUN Servers, 1553 Buses), languages (e.g. Java, XML/HTML), database managers (e.g. Oracle 8), communication tools (e.g. HTTP servers, CORBA ORBs), frameworks (e.g. CORBA, .Net, JDK 1.4), class libraries, and design patterns.
Guidelines for MBASE System and Software Architecture Description (SSAD)

- Describe how each component and connector defined in architecture structure 3.1 will be implemented (e.g. hand-coded, COTS product used, tool-generated). Refine the detailed description (i.e. interfaces, behavior, L.O.S., internal structure) of each component and connector, as appropriate, based on the implementation technology used.

- Describe how each analysis class will be implemented (e.g. language-specific classes, relational database tables) on appropriate components.

- Describe how instances of each class will be used to implement each component.

- Refine the descriptions of the behavior of the architecture and of the L.O.S. provided by the architecture base on the implementation technology.

Recommendations:

See Recommendations in Architecture Design & Analysis (SSAD 3).

UML Guideline:

Create a UML package with the stereotype <<Structure Model>> the name “Implementation Design”. Create all models described in the following sections in this package.

577 Guidelines:

Using Rose, create the UML package with the stereotype <<Structure Model>> the name “Implementation Design” in the top-level package with the name Logical View. Create subpackages as described above for simple systems. (Note: do not use Rose’s Component View or Deployment View).

4.1 Structure

Describe the architecture and the analysis classes will be implemented. Describe how each component and connector will be implemented.

- For each “hand-coded” component or connector, describe the objects and classes that will be used to implement the component or connector.14

- For components that are implemented using a COTS product, describe the values will be used for any parameters of COTS product and any tailoring that will be performed. Limited the description of COTS product to features that must be known or used by developers to implement other components in the system.

- For tool-generated components, describe inputs to the tools and the interfaces of the components that will be produced.

Describe how the implementation components and connectors are organized into layers, partitions, and subsystems. Describe which components will need to implement which analysis classes, and how the analysis classes will be realized using the technologies (e.g. programming language, database tables, web-pages) that are being used to implement the component.

---

14 We are assuming hand-coded components will be implemented using object-oriented technology, which is the most common approach at the time this version of MBASE was written.
Guidelines for MBASE System and Software Architecture Description (SSAD)

Recommendations:

The subsections of this section described below are intended to be general purpose. Tailor this section based on the size of the system, the chosen representation language, and the architectural style(s).

Model Integration Rules:

- The rationale for the Architecture must be documented in the FRD.

UML Guideline:

Either refine the contents of the “Architecture Design” package or create a UML package with the stereotype \texttt{<<systemModel>>} the name “Implementation Design” to hold all models described in the following sections.

Refining the “Architecture Design” package is simpler; but you lose the technology–independent view. Creating an “Implementation Design” package causes some redundancy; but maintains the technology–independent view, and facilitates tracing from technology–independent view to technology–specific implementations. Creating a separate package is particularly useful when creating a product–line based on the same architecture.

577 Guidelines:

Using Rose, create a UML package with the stereotype \texttt{<<systemModel>>} the name “Implementation Design” in the top–level package with the name Logical View.

4.1.1 Topology

Describe the how the components and component classes of the system are organized, e.g. layers, partitions, and/or subsystems. Refine the Topology specified in architecture design (SSAD 3.1.1) by adding any implementation–specific subsystems. (The layers and partitions should have been defined.)

Recommendations:

1. Use the representation that best suits the representation of the implementation; which may be different from that used in for architecture design.

2. If there are no changes to the topology specified during architecture design (SSAD 3.1.1) and the same representation is used, then just reference section 3.1.1.

3. For other recommendations, see Recommendations in the Topology section of architecture design (SSAD 3.1.1).

Common Pitfalls:

See Common Pitfalls in the Topology section of architecture design (SSAD 3.1.1).

Model Integration Rules:

- At LCA,
• Draft layers, partitions, and subsystems should be identified.

• Any layers or partitions specified here shall be described in the Topology section of architecture design (SSAD 3.1.1).

• Each subsystem described in the Topology section of architecture design (SSAD 3.1.1) that contains components which are high-risk or complex, shall be defined here.

• At IOC,

• Any layers, partitions, and subsystems shall be identified;

• Any layers or partitions specified here shall be described in the Topology section of architecture design (SSAD 3.1.1).

• Any subsystems specified here that are not implementation–specific shall be described in the Topology section of architecture design (SSAD 3.1.1).

577 Guidelines:

See 577 Guideline in the Topology section of architecture design (SSAD 3.1.1).

4.1.1.1 Layer X

See subsection 0 in the Topology section of architecture design (SSAD 3.1.1). Describe any changes here.

4.1.1.1.1 Partition X

See subsection 3.1.1.1.1 in the Topology section of architecture design (SSAD 3.1.1). Describe any changes here.

4.1.1.1.1.1 Subsystem X

See subsection 3.1.1.1.1.1 in the Topology section of architecture design (SSAD 3.1.1). Describe any changes here.

4.1.2 Hardware Classifier Model

Describe the kinds of hardware components that are either part of the system or on which this system will run, the actors for the system with which the system interacts, and the kinds of connectors that will be used to connect their parts.

Refine the Hardware Classifier Model specified in architecture design (SSAD 3.1.2) by describing how each hardware component and connector classifier described during architecture design is implemented, and by adding any implementation–specific hardware component and connector classifiers that are needed for support.

Representation:

See Representation in the architecture–design Hardware Classifier Model (SSAD 3.1.2).

For each implementation–specific hardware component classifier, create a subsection numbered 4.1.5.x with the name of the hardware component classifier in the header, under the section Hardware Component Classifiers (SSAD 4.1.5). The body of the subsection should describe pertinent information about the kind of hardware component.

For each implementation–specific hardware connector classifier, create a subsection numbered 4.1.6.x with the name of the hardware connector classifier in the header, under the Hardware Connector Classifiers (SSAD 4.1.6). The text of the subsection should describe the kind of hardware connector.
Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

**Trade-offs:**

See Trade-offs in the architecture–design Hardware Classifier Model (SSAD 3.1.2).

**Recommendations:**

1. Use the representation that best suits the representation of the implementation; which may be different from that used in for architecture design.

2. If there are no changes to the Hardware Classifier Model specified during architecture design (SSAD 3.1.2) and the same representation is used, then just reference section 3.1.2.

3. Create additional Static–Structure Diagram that show how each component and connector classifier defined during architecture design (SSAD 3.1.2) is implemented by one or more implementation–specific component and connector classifiers. Represent each component and connector classifier as described under Representation in this section. For each implementation–specific component and connector classifier that implementation–independent component and connector classifier, show a dependency relation with the stereotype <<trace>> from the implementation–specific to the implementation–independent component and connector classifier.$^{12,13}$

4. For other recommendations, see Recommendations in the architecture–design Hardware Classifier Model (SSAD 3.1.2).

**Common Pitfalls:**

- Confusing component instances with component classifiers.
- Including software component classifiers. (They should documented in SSAD 4.1.7)
- Including actors that are not described in the system context (SSAD 2.1).

**Model Integration Rules:**

- Each actor described here should be described the system context (SSAD 2.1).
- LCA: complete descriptions of all hardware component classifier needed to support high–risk or architecturally–significant behaviors.
- IOC: at least one instance of each hardware component classifier shown here shall appear in the Deployment Model (SSAD 4.1.4).

**577 Guidelines:**

See the 577 Guidelines in the architecture–design Hardware Classifier Model (SSAD 3.1.2).

### 4.1.3 Software Classifier Model

Describe the kinds of implementation–specific software components that are part of the system, the actors for the system with which the components interacts, and the kinds of connectors that will be used to connect their parts.

Refine the Software Classifier Model specified in architecture design (SSAD 3.1.3) by describing how each software component and connector classifier described during architecture design is implemented, and by adding any
implementation–specific software component and connector classifiers that are needed for support (e.g. frameworks).

**Representation:**

See the Representation discussion in the architecture–design Software Classifier Model (SSAD 3.1.3).

For each implementation–specific software component classifier, create a subsection numbered 4.1.7.x with the name of the software connector classifier in the header, under the Software Component Classifiers (SSAD 4.1.7). The body of the subsection should describe pertinent information about the kind of software component.

For each implementation–specific software connector classifier, create a subsection numbered 4.1.8.x with the name of the software connector classifier in the header, under the Software Connector Classifiers (SSAD 4.1.8). The text of the subsection should describe the kind of software connector.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

**Trade–offs:**

See the Trade–offs discussion in the architecture–design Software Classifier Model (SSAD 3.1.3).

**Recommendations:**

1. Use the representation that best suits the representation of the implementation; which may be different from that used in for architecture design.

2. Implementation–level stereotypes (e.g. <<ejbEntity>>) can be used to specify details about how a component classifier is implementation using supporting framework. Any implementation–level stereotypes used should ideally be defined in a standardized UML Profile (the description of the Profile should be referenced in SSAD 1.3). If you use an implementation–level stereotype that is not defined in a standard, you need to define the stereotype in either the Glossary for System Analysis and Design (SSAD 5) or as part of a UML Profile which you define. (The description of each Profile used should be referenced in SSAD 1.3)

3. If there are no changes to the Software Classifier Model specified during architecture design (SSAD 3.1.3) and the same representation is used, then just reference section 3.1.3.

4. Create additional Static–Structure Diagrams that show how each component and connector classifier defined during architecture design (SSAD 3.1.3) is implemented by one or more implementation–specific component and connector classifiers. Represent each component and connector classifier as described under Representation in this section. For each implementation–specific component and connector classifier that implementation–independent component and connector classifier, show a dependency relation with the stereotype <<trace>> from the implementation–specific to the implementation–independent component and connector classifier.12,13

5. For other recommendations, see Recommendations in the architecture–design Software Classifier Model (SSAD 3.1.3).

**Common Pitfalls:**

- Confusing component instances with component classifiers.
- Including hardware component classifiers. (They should documented in SSAD 4.1.2)
- Including actors that are not described in the system context (SSAD 2.1).
• Using undefined stereotypes on component classifiers.

Model Integration Rules:

• Each actor described here should be described the system context (SSAD 2.1).

• LCA: complete descriptions of all software component classifier needed to support high–risk or architecturally–significant behaviors.

• IOC: at least one instance of each software component classifier shown here shall appear in the Deployment Model (SSAD 4.1.4).

577 Guidelines:

See 577 Guidelines in the architecture–design Software Classifier Model (SSAD 3.1.3).

4.1.4 Deployment Model

Describe implementation–specific component and connector configuration(s) that make a working version of the system. (There may be more than one configurations, e.g. for different modes or involving different platforms.) For each configuration, describe the instances of hardware and software component classes that participate in the configuration, the allocation of software components to the hardware components, the instances of hardware connector classes that link the hardware components, and the instances software connector classes that link the software components.

Refine the Deployment Model specified in architecture design (SSAD 3.1.4) by describing how each component and connector classifier described during architecture design is implemented, and by adding any implementation–specific software components and connectors that are needed for support (e.g. frameworks).

Representation:

See the Representation discussion in the Deployment Model specified in architecture design (SSAD 3.1.4).

For each implementation–specific hardware component, create a subsection numbered 4.1.9.x with the name of the hardware component in the header, under the Hardware Components (SSAD 4.1.9). The body of the subsection should describe the hardware component.

For each implementation–specific hardware connector, create a subsection numbered 4.1.10.x with the name of the hardware connector in the header, under the Hardware Connectors (SSAD 4.1.10). The text of the subsection should describe the hardware connector.

For each implementation–specific software component, create a subsection numbered 4.1.11.x with the name of the software component in the header, under the Software Components (SSAD 4.1.11). The body of the subsection should describe the software component.

For each implementation–specific software connector, create a subsection numbered 4.1.12.x with the name of the software connector in the header, under the Software Connectors (SSAD 4.1.12). The text of the subsection should describe the software connector.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (OCD 5).

Recommendations:

1. Use the representation that best suits the representation of the implementation; which may be different from that used in for architecture design.
2. Implementation–level stereotypes (e.g. <<ejbEntity>>) can be used to specify details about how a component is implementation using supporting framework. Any implementation–level stereotypes used should ideally be defined in a standardized UML Profile (the description of the Profile should be referenced in SSAD 1.3). If you use an implementation–level stereotype that is not defined in a standard, you need to define the stereotype in either the Glossary for System Analysis and Design (SSAD 5) or as part of a UML Profile which you define. (The description of each Profile used should be referenced in SSAD 1.3)

3. For other recommendations, see Recommendations in the Deployment Model specified in architecture design (SSAD 3.1.4).

Common Pitfalls:

- Confusing component instances with component classifiers.
- Using undefined stereotypes on components.

Model Integration Rules:

- Each hardware component classifier described here should be shown in the Hardware Classifier Model (SSAD 4.1.2) and described in the section Hardware Component Classifiers (SSAD 4.1.5).
- Each hardware connector classifier described here should be shown in the Hardware Classifier Model (SSAD 4.1.2) and described in the section Hardware Connector Classifiers (SSAD 4.1.6).
- Each software component classifier described here should be shown in the Software Classifier Model (SSAD 4.1.3) and described in the section Software Component Classifiers (SSAD 4.1.7).
- Each software connector classifier described here should be shown in the Software Classifier Model (SSAD 4.1.3) and described in the section Software Connector Classifiers (SSAD 4.1.8).
- Each actor described here should be described the system context (SSAD 2.1).
- At LCA, the System Deployment Model shall show the final configuration of implementation–specific hardware and software that participate in high–risk behaviors, and the preliminary allocation of all other components.
- At IOC, the System Deployment Model shall show the final configuration of all implementation–specific components.

577 Guidelines:

See 577 Guidelines in the Deployment Model specified in architecture design (SSAD 3.1.4).

4.1.5 Hardware Component Classifiers

For each implementation–specific hardware component classifier shown in the figure(s) shown in the Hardware Classifier Model (SSAD 4.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware component classifier in the header.

Representation:

The description of implementation–specific hardware components classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations. For example, if you are describing a system where the hardware is not being developed as part
of the system (e.g. a web application), it is usually sufficient to just describe the purpose of each hardware component and the L.O.S. that the hardware is expected to provide.

577 Guidelines:

Your system is probably simple enough, that you need only fill in the following sections for each component.

- 4.1.5.1.1 Purpose
- 4.1.5.1.5 L.O.S. Goals (needed to determine some system goals)

If you decide more detail is needed, create a UML package, with the stereotype &lt;&lt;node&gt;&gt; and the name “Component Classifier Name ID”, in the package that holds the classifier that represents this hardware component classifier.

4.1.5.1 Component Classifier X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill–in the details for the following sections.

4.1.5.1.1 Purpose

Describe the purpose of this class of component.

4.1.5.1.2 Interface(s)

Describe the visible features of the component classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

Representation:

See Representation for Interface(s) of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.2).

For each feature or set of features, create a subsection numbered 4.1.5.1.2.x and the name of the hardware feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Model Integration Rules:

- Each interface feature described here, should be used in the component’s behavior description (SSAD 4.1.7.1.4).
- At LCA: a preliminary set of interfaces for the component shall be described.
- At IOC: the final set of interfaces for the component shall be described.

577 Guidelines:

Your system is probably simple enough, that you are unlikely to need this detailed of a description for your hardware. If you decide this level of detail is needed, see 577 Guidelines for Interface(s) of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.2).
4.1.5.1.2.1 Feature or Feature Set X

See Feature or Feature Set X for Interface(s) of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.2).

4.1.5.1.3 Parameters

Describe any parameters of the hardware component classifier. The parameters need to be set when an instance of the hardware component classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other components.

Representation:

See Representation for Parameters of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.3).

Model Integration Rules:

- At LCA: a preliminary set of parameters for the component, if any exist, should be described.
- At IOC: the final set of parameters for the component shall be described. (If none, say so.)

577 Guidelines:

Your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware. If you decide this level of detail is needed, see 577 Guidelines for Parameters of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.3).

4.1.5.1.4 Behavior

Describe behavior of the instances of this component classifier.

Representation:

The representation of behavior is depends on the language used to represent it. The following subsection work for UML–based architecture descriptions and may work for ADL’s.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

- At LCA: a draft behavior for the component shall be described if the component is high–risk.
- At IOC: the behavior for the component shall be described if the component is included in the build.

UML Guidelines:

Describe the processes of this component classifier, including how it works with the other components and the system’s actors, and how it uses the artifacts and information; and if significant, how the component’s behavior depends on its mode.

577 Guidelines:

Your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware.
4.1.5.1.4.1 Processes

Describe the processes of this component classifier. For each process, identify which other components and actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by this component and each actor during the process.

Representation:

Create a Use–Case Model that describes the component’s processes in the same way that a model was created to describe the system’s processes in SSAD 4.3. Represent the model as a package with the stereotype <<use–case model>> with the package name “Component Classifier Name Processes”.

- Create one or more Use–Case Diagrams
- For each process, create a Use–Case Description, and an Activity Diagram.

For each process, create a subsection numbered 4.1.5.1.4.1.x (see example section below) with the name of the process in the header. Each subsection should provide the information shown in the Figure 4 and an Activity Diagram.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. If the component whose behavior is being described is the implementation of a component defined during architecture design, then

   - If there are no implementation–specific processes, you need not create new diagram(s); just insert a reference to the diagram(s) created for the architecture–design component.
   - If there are implementation–specific processes, create Use–Case Diagram(s) that show the implementation–specific processes and their relations.
   - For each use–case that was described during architecture–design,
     - Insert a reference to the description created for the architecture–design. (The details should be the same.)
     - Create an Activity Diagram only if there is implementation–specific behavior.

2. For additional recommendations, see Recommendations for Processes of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.4.1).

Common Pitfalls:

- Spending too much time describing the processes in detailed for low risk components. This problem is more common for Simple Systems and small Composite Systems.
- For additional pitfalls, see Common Pitfalls for system’s System Capabilities (SSAD 4.3).

Model Integration Rules:

- Each operation defined in the component’s interface (SSAD 4.1.5.1.2) must either be represented by a use–case or represented as an action in a use–case’s Activity Diagram.
• Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).

• Each component in a process description should be defined in the architecture of the system (SSAD 4.1).

• Each classes used or produced in a process description should be defined in the classes for the system’s implementation (SSAD 4.1.13).

4.1.5.1.4.1.1 Process X

Create a section at this level for each process of the component. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.

4.1.5.1.4.2 Modes of Operation

Describe the modal behavior of the component instances, i.e. how the behavior of the component depends on the mode it is in. Describe the high–level states of the component, the events that cause the component to change modes, and how the processing is different in each mode.

Representation:

Create a State Model that describes the connector’s modes in the same way that a model was created to describe the system’s modes in SSAD 2.3.2. Represent the model as a package with the stereotype <<state model>> with the package name “Modes of Component Classifier Name”. Create one or more Statecharts that show the modes of component and the events that cause mode changes.

For each mode, create a subsection numbered 4.1.5.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list component’s processes (SSAD 4.1.5.1.4.1) that are available in that mode (see Table 14 on page 6). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode–specific behavior).

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right–click on the <<node>> classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Model” that is attached to the <<node>> classifier, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

Recommendations:

1. If the component whose behavior is being described is the implementation of a component defined during architecture design, then

   • If there are no implementation–specific modes, you need not create new diagram(s); just insert are reference to the diagram(s) created for the architecture–design component.

   • If there are no implementation–specific modes or implementation–specific processes defined in SSAD 4.1.5.1.4.1, you need not create new table of processes in modes, just insert are reference to the table created for the architecture–design component.

2. For additional recommendations, see Recommendations for system’s Modes of Operation (SSAD 2.3.2).
Common Pitfalls:

- Spending too much time describing the modes in detail for low–risk components or components that do not have significant modal behavior. This problem is more common for Simple Systems and small Composite Systems.

- For additional pitfalls, see Common Pitfalls for system’s Modes of Operation (SSAD 2.3.2).

Model Integration Rules:

- Each component process (SSAD 4.1.5.1.4.1) should be listed as available in one or more modes.

- Each process listed as available in one or more modes should appear in described in SSAD 4.1.5.1.4.1.

4.1.5.1.4.2.1 Mode X

Create a section at this level for each mode of the component. The header of this section should be the name of the mode and its unique designator, if you have assigned an identifier and it is different from the name. Describe the mode (e.g. for the airplane’s taxi mode, “when in this mode, the airplane is moving on the ground, usually to or from a gate”).

List the component’s processes (SSAD 4.1.5.1.4.1) that are available in this mode (see Table 14). For each process, describe any effect of the mode.

4.1.5.1.5 L.O.S. Goals

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component classifier. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a component classifier defines the goals for all instances of this classifier, and the goals that an implementation of this component classifier must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the system’s goals (i.e. the system goals will be met its goals if all connectors and components achieve their goals).

Representation:

For each system goal that applies to this component class fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its own value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

See Recommendations for L.O.S. Goals of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.5).

Common Pitfalls:

- Not satisfying the M.R.S. criteria.

- Specifying values that do not apply to all instances.

- Specifying values for a goal that exceeds values, or is outside the range of values, specified for the architecture–design component that this component implements.
Model Integration Rules:

- Each system L.O.S. Goals (OCD 4.4) should be described here.
- Each process listed here, should be described in the component’s Processes (SSAD 4.1.5.1.4.1).
- Each mode listed here, should be described in the system’s Modes of Operation (SSAD 4.1.5.1.4.2).
- Each class listed here, should be described in the implementation Implementation Classes (SSAD 4.1.13).

4.1.5.1.6 Constraints

Describe any constraints on the use and implementation of this component that are not captured in other sections of the component description. The following paragraphs describe some typical constraints.

- Rules that the component must implement in support of rules that the system must satisfy;
- Constraints imposed on component’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on component’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the component’s design and implementation, without getting into implementation details.

Representation:

See Representation for Constraints of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.6).

Trade-offs:

See Trade-offs in section OCD 3.5(current organization rules).

Recommendations:

1. To facilitate traceability, assign each rule a unique number (e.g. Component-Rule-01 or CR-01).
2. For additional recommendations, see Recommendations in section OCD 3.5(current organization rules).

Common Pitfalls:

- Describing a rule that is already captured in another view of the component (e.g. modes of operation (SSAD 4.1.5.1.4.2)).

Model Integration Rules:

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each process described in a constraint, should be described in the component classifier’s Processes (SSAD 4.1.5.1.4.1).
- Each mode described in a constraint, should be described in the component classifier’s Modes of Operation (SSAD 4.1.5.1.4.2).
• Each artifact or information classifier described in a constraint, should be described in the architecture’s Implementation Classes (SSAD 4.1.13).

• IOC: Each component rule should be implemented by one or more elements described in either this component’s Internal Architecture (SSAD 4.1.6.1.7), or its Implementation Design (SSAD 4).

4.1.5.1.7 Internal Architecture

Describe the architecture of this hardware component classifier that is independent of the implementation technology. Describe the subcomponents of this component, what they are expected to do, how they are connected, and what they communicate.

Representation:

See Representation for Internal Architecture of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.7).

Common Pitfalls:

Spending time describing the internal architecture for components that will not be implemented by this project. (This problem is more common for Simple Systems and small Composite Systems.) If the component is low risk, consider deferring its details until Implementation Design (SSAD 4).

577 Guidelines:

See 577 Guideline for Internal Architecture of architecture–design Hardware Component Classifiers (SSAD 3.1.5.1.7).

4.1.6 Hardware Connector Classifiers

For each implementation-specific hardware connector classifier defined in the figure(s) shown in the Hardware Classifier Model (SSAD 4.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware connector classifier in the header.

Representation:

The description of hardware connector classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations. For example, if you are describing a system where the hardware is not being developed as part of the system (e.g. a web application), it is usually sufficient to just describe the purpose of each hardware connector and the L.O.S. that the hardware is expected to provide.

The subsections of this section are intended to support a wide range of systems, architecture languages, and architecture styles.

577 Guidelines:

Your system is probably simple enough, that you need only fill in the following sections for each connector.

• 4.1.6.1.1 Purpose

• 4.1.6.1.5 L.O.S. Goals (needed to determine some system goals)

If you decide this level of detail is needed, create a UML package with the stereotype <<connector>> the name “Connector Classifier Name ID” in the package that holds the corresponding association with the stereotype <<connector>>.
4.1.6.1 **Connector Classifier X**

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill—in the details for the following sections.

### 4.1.6.1.1 Purpose

Describe the purpose of this class of connector.

### 4.1.6.1.2 Interface(s)

Describe the visible features of the connector classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

**Representation:**

See Representation for Interface(s) of architecture—design Hardware Connector Classifiers (SSAD 3.1.6.1.2).

For each feature or set of features, create a subsection numbered 4.1.6.1.2.x and the name of the hardware feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

**UML Guidelines:**

See UML Guidelines for Interface(s) of architecture—design Hardware Connector Classifiers (SSAD 3.1.6.1.2).

**577 Guidelines:**

Your system is probably simple enough, that you are unlikely to need to describe the interfaces of hardware connector classifiers.

If you decide this level of detail is needed, follow the UML Guidelines.

#### 4.1.6.1.2.1 Feature or Feature Set X

See Feature or Feature Set X for Interface(s) of architecture—design Hardware Connector Classifiers (SSAD 3.1.6.1.2).

### 4.1.6.1.3 Parameters

Describe any parameters of the hardware connector classifier. The parameters need to be set when an instance of the connector classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other connectors.

**Representation:**

See Representation for Parameters of architecture—design Hardware Connector Classifiers (SSAD 3.1.6.1.3).

**Model Integration Rules:**

- At LCA: a preliminary set of parameters for the connector, if any exist, shall be described.
- At IOC: the final set of parameters for the connector shall be described. (If none, say so.)
UML Guidelines:

See UML Guidelines for Parameters of architecture–design Hardware Connector Classifiers (SSAD 3.1.6.1.3).

577 Guidelines:

Your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware. If you decide this level of detail is needed, follow the UML Guidelines.

4.1.6.1.4 Behavior

Describe behavior of the instances of this connector classifier.

Representation:

Describe the behavior of the connector classifier using an Architecture Description Language (ADL). The form of the specification will depend on the language used. (See the ADL’s manual for details.)

The following subsection work for UML–based architecture descriptions and may work for other ADL’s.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

For systems where the hardware is not being developed as part of the project, it is usually unnecessary to specify its behavior.

Model Integration Rules:

- At LCA: a draft behavior for the connector shall be described if the connector is high–risk.
- At IOC: the behavior for the connector shall be described if the connector is included in the build.

UML Guidelines:

Describe the processes of this connector classifier, including how it works with the other connectors and the components; and if significant, how the connector’s behavior depends on its mode.

577 Guidelines:

Your systems are simple enough, that you are unlikely to need this detailed of a description for your hardware.

4.1.6.1.4.1 Processes

Describe the processes of this connector classifier. For each process, identify which other connectors and actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by this connector and each actor during the process.

Representation:

Create a Use–Case Model that describes the connector’s processes in the same way that a model was created to describe the system’s processes in SSAD 4.3. Represent the model as a package with the stereotype <<use–case model>> with the package name “Connector Classifier Name Processes”.

- Create one or more Use–Case Diagrams
- For each process, create a Use–Case Description, and an Activity Diagram.
For each process, create a subsection numbered 4.1.6.1.4.1.x (see example section below) with the name of the process in the header. Each subsection should provide the information shown in the Figure 4 and an Activity Diagram.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

1. If the connector whose behavior is being described is the implementation of a connector defined during architecture design, then
   
   - If there are no implementation–specific processes, you need not create new diagram(s); just insert a reference to the diagram(s) created for the architecture–design connector.
   
   - If there are implementation–specific processes, create Use–Case Diagram(s) that show the implementation–specific processes and their relations.
   
   - For each use–cases that was described during architecture–design,
     
     - Insert a reference to the description created for the architecture–design. (The details should be the same.)
     
     - Create an Activity Diagram only if there is implementation–specific behavior.

2. For additional recommendations, see Recommendations for Processes of architecture–design Hardware Connector Classifiers (SSAD 3.1.6.1.4.1).

**Common Pitfalls:**

For additional pitfalls, see Common Pitfalls for system’s System Capabilities (SSAD 4.3).

**Model Integration Rules:**

- Each operation defined in the connector’s interface (SSAD 4.1.6.1.2) must either be represented by a use–case or represented as an action in a use–case’s Activity Diagram.

- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).

- Each component in a process description should be defined in the architecture of the system (SSAD 4.1).

- Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information for the system’s architecture (SSAD 4.1.13).

**4.1.6.1.4.1.1 Process X**

Create a section at this level for each process of the connector. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.
4.1.6.1.4.2 Modes of Operation

Describe the modal behavior of the connector instances, i.e. how the behavior of the connector depends on the mode it is in. Describe the high–level states of the connector, the events that cause the connector to change modes, and how the processing is different in each mode.

Representation:

As in SSAD 2.3.2, create a State Model. Represent the model as a package with the stereotype <<state model>> with the package name “Modes of Connector Classifier Name”. Create one or more Statecharts that show the modes of connector and the events that cause mode changes.

For each mode, create a subsection numbered 4.1.6.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list connector’s processes (SSAD 4.1.6.1.4.1) that are available in that mode (see Table 16 on page 6). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode–specific behavior).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right–click on the connector’s association class, which you should have created when you described the connector’s interfaces (SSAD 4.1.6.1.2),15 in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Modelx” that is attached to the connector’s association class, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

Recommendations:

1. If the connector whose behavior is being described is the implementation of a connector defined during architecture design, then

   - If there are no implementation–specific modes, you need not create new diagram(s); just insert are reference to the diagram(s) created for the architecture–design connector.

   - If there are no implementation–specific modes or implementation–specific processes defined in SSAD 4.1.5.1.4.1, you need not create new table of processes in modes, just insert are reference to the table created for the architecture–design connector.

2. For additional recommendations, see Recommendations for system’s Modes of Operation (SSAD 2.3.2).

Common Pitfalls:

See Common Pitfalls for system’s Modes of Operation (SSAD 2.3.2).

Model Integration Rules:

   - Each connector process (SSAD 4.1.6.1.4.1) should be listed as available in one or more modes.

   - Each process listed as available in one or more modes should appear in described in SSAD 4.1.6.1.4.1.

15 If you are describing the behavior of a connector, then you should have described one or more interfaces.
4.1.6.1.4.2.1 Mode X

Create a section at this level for each mode of the connector. The header of this section should be the name of the mode and its unique designator, if you have assigned an identifier and it is different from the name. Describe the mode (e.g. for the airplane’s taxi mode, “when in this mode, the airplane is moving on the ground, usually to or from a gate”).

List the connector’s processes (SSAD 4.1.6.1.4.1) that are available in this mode (see Table 16). For each process, describe any effect of the mode.

4.1.6.1.5 L.O.S. Goals

Describe how the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a connector classifier defines the goals for all instances of this classifier, and the goals that an implementation of this connector classifier must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the system’s goals (i.e. the system goals will be meet its goals if all connectors and components achieve their goals).

Representation:

For each system goal that applies to this connector fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its on value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

Recommendations:

See Recommendations for L.O.S. Goals of architecture–design Hardware Connector Classifiers (SSAD 3.1.6.1.5).

Common Pitfalls:

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.
- Specifying values for a goal that exceeds values, or is outside the range of values, specified for the architecture–design connector that this connector implements.

Model Integration Rules:

- Each connector L.O.S. Goals (OCD 4.4) should be described here.
- Each process listed here, should be described in the connector’s Processes (SSAD 4.1.6.1.4.1).
- Each mode listed here, should be described in the system’s Modes of Operation (SSAD 4.1.6.1.4.2).
- Each class listed here, should be described in the implementation Implementation Classes (SSAD 4.1.13).

4.1.6.1.6 Constraints

Describe any constraints on the use and implementation of this connector that are not captured in other sections of the connector description. The following paragraphs describe some typical constraints.

- Rules that the connector must implement in support of rules that the system must satisfy;
Guidelines for MBASE System and Software Architecture Description (SSAD)

- Constraints imposed on connector’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;

- Constraints imposed on connector’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the connector’s design and implementation, without getting into implementation details.

**Representation:**

See Representation for Constraints of architecture–design Hardware Connector Classifiers (SSAD 3.1.6.1.6).

**Trade-offs:**

See Trade–offs in section OCD 3.5(current organization rules).

**Recommendations:**

1. To facilitate traceability, assign each rule a unique number (e.g. Connector-Rule-01).

2. For additional recommendations, see Recommendations in section OCD 3.5(current organization rules).

**Common Pitfalls:**

- Describing a rule that is already captured in another view of the connector (e.g. modes of operation (SSAD 4.1.6.1.4.2)).

**Model Integration Rules:**

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).

- Each actor described in a rule should be defined in the system context (SSAD 2.1).

- Each process described in a constraint, should be described in the connector’s Processes (SSAD 4.1.6.1.4.1).

- Each mode described in a constraint, should be described in the system’s Modes of Operation (SSAD 4.1.6.1.4.2).

- Each class described in a constraint, should be described in the implementation Implementation Classes (SSAD 4.1.13).

- IOC: Each connector rule should be implemented by one or more elements described in this connector’s Internal Architecture (SSAD 4.1.5.1.7).

**4.1.6.1.7 Internal Architecture**

Describe the architecture of this hardware connector classifier that is independent of the implementation technology. Describe the subconnectors of this connector, what they are expected to do, how they are connected, and what they communicate.

**Representation:**

See Representation for Internal Architecture of architecture–design Hardware Connector Classifiers (SSAD 3.1.6.1.7).
577 Guidelines:

See 577 Guidelines for Internal Architecture of architecture–design Hardware Connector Classifiers (SSAD 3.1.6.1.7).

4.1.7 Software Component Classifiers

For each implementation–specific software component classifier defined in the figure(s) shown in the Software Classifier Model (SSAD 4.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the software component classifier in the header.

Representation:

The description of software components classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

Common Pitfalls:

For Simple Systems or Composite Systems, there is a risk of spending too much time describing the behavior (4.1.5.1.4) or the internal architecture (4.1.6.1.7) of a component in detailed.

Model Integration Rules:

- At LCA: Each architecture–design components and interfaces of components described in (SSAD 3.1.7) that are high–risk shall have an implementation defined here.
- At IOC: All architecture–design components and interfaces of components described in (SSAD 3.1.7) that are high–risk shall be have implementation defined here.

577 Guidelines:

Create a UML package, with the stereotype <<component>> and the name “Component Classifier Name ID”, in the package that holds the classifier that represents this software component classifier.

4.1.7.1 Component Classifier X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill–in the details for the following sections.

4.1.7.1.1 Purpose

Describe the purpose of this class of component.

4.1.7.1.2 Interface(s)

Describe the visible features of the component classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

Representation:

See Representation for Interface(s) of architecture–design Software Component Classifiers (SSAD 3.1.7.1.2).

For each feature or set of features, create a subsection numbered 4.1.7.1.2.x and the name of the software feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.
Guidelines for MBASE System and Software Architecture Description (SSAD)

Model Integration Rules:

- Each interface feature described here, should be used in the component’s behavior description (SSAD 4.1.7.1.4).
- At LCA:
  - If the component is high-risk, then a set of interfaces for the component shall be described.
  - If the component is not high-risk, then a draft set of interfaces for the component should be described.
  - At IOC: the final set of interfaces for the component shall be described.

577 Guidelines:

See 577 Guidelines for Interface(s) of architecture–design Software Component Classifiers (SSAD 3.1.7.1.2).

4.1.7.1.2.1 Feature or Feature Set X

See Feature or Feature Set X for Interface(s) of architecture–design Software Component Classifiers (SSAD 3.1.7.1.2).

4.1.7.1.3 Parameters

Describe any parameters of the software component classifier. The parameters need to be set when an instance of the software component classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other components.

Representation:

See Representation for Parameters of architecture–design Software Component Classifiers (SSAD 3.1.7.1.3).

Model Integration Rules:

- At LCA: a preliminary set of parameters for the component, if any exist, should be described.
- At IOC: the final set of parameters for the component shall be described. (If none, say so.)

577 Guidelines:

See 577 Guidelines for Parameters of architecture–design Software Component Classifiers (SSAD 3.1.7.1.3).

4.1.7.1.4 Behavior

Describe behavior of the instances of this component classifier.

Representation:

The representation of behavior is depends on the language used to represent it. The following subsection work for UML–based architecture descriptions and may work for ADL’s.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).
Model Integration Rules:

- At LCA: a draft behavior for the component shall be described if the component is high-risk.
- At IOC: the behavior for the component shall be described if the component is included in the build.

UML Guidelines:

Describe the processes of this component classifier, including how it works with the other components and the system’s actors, and how it uses the artifacts and information; and if significant, how the component’s behavior depends on its mode.

4.1.7.1.4.1 Processes

Describe the processes of this component classifier. For each process, identify which other components and actors participate in the process, and which artifacts and information are inspected, manipulated, or produced by the process; and describe at a high level the actions performed by this component and each actor during the process.

Representation:

Create a Use-Case Model that describes the component’s processes in the same way that a model was created to describe the system’s processes in SSAD 4.3. Represent the model as a package with the stereotype <<use-case model>> with the package name “Component Classifier Name Processes”.

- Create one or more Use-Case Diagrams
- For each process, create a Use-Case Description, and an Activity Diagram.

For each process, create a subsection numbered 4.1.7.1.4.1.x (see example section below) with the name of the process in the header. Each subsection should provide the information shown in the Figure 4 and an Activity Diagram.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. If the component whose behavior is being described is the implementation of a component defined during architecture design, then

   - If there are no implementation–specific processes, you need not create new diagram(s); just insert a reference to the diagram(s) created for the architecture–design component.
   - If there are implementation–specific processes, create Use–Case Diagram(s) that show the implementation–specific processes and their relations.
   - For each use–cases that was described during architecture–design,
     - Insert a reference to the description created for the architecture–design. (The details should be the same.)
     - Create an Activity Diagram only if there is implementation–specific behavior.

2. For additional recommendations, see Recommendations for Processes of architecture–design Software Component Classifiers (SSAD 3.1.7.1.4.1).
Common Pitfalls:

- Spending too much time describing the processes in detailed for low risk components. This problem is more common for Simple Systems and small Composite Systems.

- For additional pitfalls, see Common Pitfalls for system’s System Capabilities (SSAD 4.3).

Model Integration Rules:

- Each operation defined in the component’s interface (SSAD 4.1.7.1.2) must either be represented by a use–case or represented as an action in a use–case’s Activity Diagram.

- Each actor in a process description should be defined in the structure of the context of the system (SSAD 2.1).

- Each component in a process description should be defined in the architecture of the system (SSAD 4.1).

- Each artifact or information classifier used or produced in a process description should be defined in the artifacts and information for the system’s architecture (SSAD 4.1.13).

4.1.7.1.4.1.1 Process X

Create a section at this level for each process of the component. The header of this section should be the name of the process and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.

4.1.7.1.4.2 Modes of Operation

Describe the modal behavior of the component instances, i.e. how the behavior of the component depends on the mode it is in. Describe the high–level states of the component, the events that cause the component to change modes, and how the processing is different in each mode.

Representation:

Create a State Model that describes the component’s modes in the same way that a model was created to describe the system’s modes in SSAD 2.3.2. Represent the model as a package with the stereotype <<state model>> with the package name “Modes of Component Classifier Name”. Create one or more Statecharts that show the modes of component and the events that cause mode changes.

For each mode, create a subsection numbered 4.1.7.1.4.2.x (see example section below) with the name of the mode in the header. Describe the mode, and list component’s processes (SSAD 4.1.7.1.4.1) that are available in that mode (see Table 14 on page 6). For each process, describe any effect of the mode (e.g., a process may have a higher L.O.S. in one mode then in another, the process may have mode–specific behavior).

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right–click on the <<component>> classifier in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Modelx” that is attached to the <<component>> classifier, and a diagram named “NewDiagram” in the model. Rename the State Model to “Modes of Operation” and the diagram to “Top Level”.

577 Guidelines:
Guidelines for MBASE System and Software Architecture Description (SSAD)

Recommendations:

1. If the component whose behavior is being described is the implementation of a component defined during architecture design, then

   - If there are no implementation-specific modes, you need not create new diagram(s); just insert a reference to the diagram(s) created for the architecture-design component.

   - If there are no implementation-specific modes or implementation-specific processes defined in SSAD 4.1.5.1.4.1, you need not create new table of processes in modes, just insert a reference to the table created for the architecture-design component.

2. For additional recommendations, see Recommendations for system’s Modes of Operation (SSAD 2.3.2).

Common Pitfalls:

- Spending too much time describing the modes in detail for low-risk components or components that do not have significant modal behavior. This problem is more common for Simple Systems and small Composite Systems.

- For additional pitfalls, see Common Pitfalls for system’s Modes of Operation (SSAD 2.3.2).

Model Integration Rules:

- Each component process (SSAD 4.1.7.1.4.1) should be listed as available in one or more modes.

- Each process listed as available in one or more modes should appear in described in SSAD 4.1.7.1.4.1.

4.1.7.1.4.2.1 Mode X

Create a section at this level for each mode of the component. The header of this section should be the name of the mode and its unique designator, if you have assigned an identifier and it is different from the name. Describe the mode (e.g., for the airplane’s taxi mode, “when in this mode, the airplane is moving on the ground, usually to or from a gate”).

List the component’s processes (SSAD 4.1.7.1.4.1) that are available in this mode (see Table 14). For each process, describe any effect of the mode.

4.1.7.1.5 L.O.S. Goals

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component classifier. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a component classifier defines the goals for all instances of this classifier, and the goals that an implementation of this component classifier must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the system’s goals (i.e., the system goals will be meet its goals if all connectors and components achieve their goals).

Representation:

For each system goal that applies to this component class fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its own value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.
Guidelines for MBASE System and Software Architecture Description (SSAD)

Recommendations:

See Recommendations for L.O.S. Goals of architecture–design Software Component Classifiers (SSAD 3.1.7.1.5).

Common Pitfalls:

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.
- Specifying values for a goal that exceeds values, or is outside the range of values, specified for the architecture–design component that this component implements.

Model Integration Rules:

- Each component L.O.S. Goals (OCD 4.4) should be described here.
- Each process listed here, should be described in the component’s Processes (SSAD 4.1.7.1.4.1).
- Each mode listed here, should be described in the system’s Modes of Operation (SSAD 4.1.7.1.4.2).
- Each class listed here, should be described in the implementation Implementation Classes (SSAD 4.1.13).

4.1.7.1.6 Constraints

Describe any constraints on the use and implementation of this component that are not captured in other sections of the component description. The following paragraphs describe some typical constraints.

- Rules that the component must implement in support of rules that the system must satisfy;
- Constraints imposed on component’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on component’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the component’s design and implementation, without getting into implementation details.

Representation:

See Representation for Constraints of architecture–design Software Component Classifiers (SSAD 3.1.7.1.6).

Trade–offs:

See Trade–offs in section OCD 3.5(current organization rules).

Recommendations:

1. To facilitate traceability, assign each rule a unique number (e.g. Component-Rule-01 or CR-01).

2. For additional recommendations, see Recommendations in section OCD 3.5(current organization rules).
Common Pitfalls:

- Describing a rule that is already captured in another view of the component (e.g. modes of operation (SSAD 4.1.7.1.4.2)).

Model Integration Rules:

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each process described in a constraint, should be described in the component classifier’s Processes (SSAD 4.1.7.1.4.1).
- Each mode described in a constraint, should be described in the component classifier’s Modes of Operation (SSAD 4.1.7.1.4.2).
- Each artifact or information classifier described in a constraint, should be described in the architecture’s Implementation Classes (SSAD 4.1.13).
- IOC: Each component rule should be implemented by one or more elements described in either this component’s Internal Architecture (SSAD 4.1.7.1.7), or its Implementation Design (SSAD 4).

4.1.7.1.7 Internal Architecture

Describe the architecture of this software component classifier that is independent of the implementation technology.

- For components that represent lowest–level modular, deployable, and replaceable part of a system (e.g. the representation of an executable, a link–library, a Java Been), describe the objects and classes that are used to create the component.
- For other components, describe the subcomponents of this component, what they are expected to do, how they are connected, and what they communicate.

Representation:

Apply the same guidelines to describe the architecture of this component as was used to describe for the system architecture in section Implementation Design (SSAD 4), i.e. create subsections: “Structure”, “Implementation Classes”, “Behavior”, “L.O.S.”, and “Patterns & Frameworks”.

For components that represent lowest–level modular, deployable, and replaceable part of a system, the internal architecture consists of the objects and their classes that are used to build the component.

- Create a Static–Structure Model that shows the objects and classes that are used to create the component. Create one or more Static–Structure Diagrams that show the resident classes and interfaces that implement this component; the attributes of each resident class; the operations of each resident class; the classes or interfaces defined in other components which are used by the resident classes; and the relations among the classes and interfaces.

Create one or more Collaboration Diagrams show the particular configurations of the class instances (“objects”), possibly for a specific purpose, that implement this component. If two instances interact in this configuration, then show a link connecting the two instances.

Each Static–Structure Diagram and Collaboration Diagram should be accompanied by a brief description of its purpose.
For other components, create a Structure Model that describes the sub-components and their classes as defined in SSAD 4.1

Recommendations:

1. For a System of System, instead of documenting the internal architecture here, include a reference to the project documents (e.g. OCD, SSAD) for the project that is responsible for creating the subsystem represented by this component.

2. If the describing the objects and classes that are used to build the component, create additional Static–Structure Diagrams that show which analysis classes defined during Architecture Design & Analysis (SSAD 3.2) is implemented by one or more implementation classes in this component. Represent each analysis class as described in Analysis Classes (SSAD 3.2), and each implementation class as in this section. For each class that implements an analysis class, show a dependency relation with the stereotype <<trace>> from the implementation class to the analysis class.12,13

3. Implementation–level stereotypes can be used to specify details about how a component classifier or a class is implementation using supporting framework (e.g. <<JavaBean>> for a component implemented as a Java Bean, or <<JSP>> for class that represents a Java Server Page). Any implementation–level stereotypes used should ideally be defined in a standardized UML Profile (the description of the Profile should be referenced in SSAD 1.3). If you use an implementation–level stereotype that is not defined in a standard, you need to define the stereotype in either the Glossary for System Analysis and Design (SSAD 5) or as part of a UML Profile which you define. (The description of each Profile used should be referenced in SSAD 1.3)

577 Guidelines:

Create a UML package with the stereotype <<Structure Model>> the name “Architecture” in the package representing this component (i.e. the one with stereotype <<component>> and the name “Component Classifier Name ID”) to hold the internal architecture description.

4.1.8 Software Connector Classifiers

For each software connector classifier defined in the figure(s) shown in the Software Classifier Model (SSAD 4.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the software connector classifier in the header.

Representation:

The description of software connector classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

Software Connectors are a relatively new concept in software architectures. Few ADL’s represent software connectors in any details.

The subsections of this section are intended to support a wide range of systems, architecture languages, and architecture styles.

Model Integration Rules:

- At LCA: Connectors and interfaces of connectors that are high–risk shall be described in sufficient detail to support evaluation of correctness, and have a draft implementation design (SSAD 4.1).
• At IOC: All connectors and interfaces of connectors in the build shall be described in sufficient detail to support evaluation of correctness, and have a designed implementation (SSAD 4.1).

UML Guidelines:

The current definition of UML [OMG 2001] does not define software connectors. If you are using the UML standard without extension, delete this section.

577 Guidelines:

Since we are using UML [OMG 2001] without extensions, delete this section.

4.1.8.1 Connector Classifier X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill–in the details for the following sections.

4.1.8.1.1 Purpose

Describe the purpose of this class of connector.

4.1.8.1.2 Interface(s)

Describe the visible features of the connector classifier.

What features can be specified in an interface depend on your architecture style and language (inc. UML). Some common features include operations, attributes, and subcomponents.

Representation:

See Representation for Interface(s) of architecture–design Software Connector Classifiers (SSAD 3.1.8.1.2).

For each feature or set of features, create a subsection numbered 4.1.8.1.2.x and the name of the software feature or set of features in the header. The body of the subsection should describe pertinent information about the feature or set of features.

Model Integration Rules:

• Each interface feature described here, should be used in the connector’s behavior description (SSAD 4.1.7.1.4).

• At LCA: a preliminary set of interfaces for the connector shall be described.

• At IOC: the final set of interfaces for the connector shall be described.

4.1.8.1.2.1 Feature or Feature Set X

See Feature or Feature Set X for Interface(s) of architecture–design Software Connector Classifiers (SSAD 3.1.8.1.2).

4.1.8.1.3 Parameters

Describe any parameters of the software connector classifier. The parameters need to be set when an instance of the connector classifier is created.

What parameters can be specified, if any, depend on your architecture style and language (inc. UML). Some common parameters include values, objects, classifiers, operations, and other connectors.
Representation:

See Representation for Parameters of architecture–design Software Connector Classifiers (SSAD 3.1.8.1.3).

Model Integration Rules:

- At LCA: a preliminary set of parameters for the connector, if any exist, shall be described.
- At IOC: the final set of parameters for the connector shall be described. (If none, say so.)

4.1.8.1.4 Behavior

Describe behavior of the instances of this connector classifier.

Representation:

Describe the behavior of the connector classifier using an Architecture Description Language (ADL). The form of a specification using an ADL will depend on the language used. (See the ADL’s manual for details.)

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

4.1.8.1.5 L.O.S. Goals

Describe how the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a connector classifier defines the goals for all instances of this classifier, and the goals that an implementation of this connector classifier must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the system’s goals (i.e. the system goals will be meet its goals if all connectors and components achieve their goals).

Representation:

For each system goal that applies to this connector fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its on value, then specify that the “value is specific to the component instance”. Optionally specify a default value or a limiting value.

Recommendations:

See Recommendations for L.O.S. Goals of architecture–design Software Connector Classifiers (SSAD 3.1.8.1.5).

Common Pitfalls:

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.
- Specifying values for a goal that exceeds values, or is outside the range of values, specified for the architecture–design connector that this connector implements.

Model Integration Rules:

- Each connector L.O.S. Goals (OCD 4.4) should be described here.
- Each class listed here, should be described in the implementation Implementation Classes (SSAD 4.1.13).
4.1.8.1.6 Constraints

Describe any constraints on the use and implementation of this connector that are not captured in other sections of the connector description. The following paragraphs describe some typical constraints.

- Rules that the connector must implement in support of rules that the system must satisfy;
- Constraints imposed on connector’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on connector’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the connector’s design and implementation, without getting into implementation details.

**Representation:**

See Representation for Constraints of architecture–design Software Connector Classifiers (SSAD 3.1.8.1.6).

**Trade–offs:**

See Trade–offs in section OCD 3.5(current organization rules).

**Recommendations:**

1. To facilitate traceability, assign each rule a unique number (e.g. Connector-Rule-01).

2. For additional recommendations, see Recommendations in section OCD 3.5(current organization rules).

**Common Pitfalls:**

- Describing a rule that is already captured in another view of the connector (e.g. behavior (SSAD 4.1.8.1.4)).

**Model Integration Rules:**

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each actor described in a rule should be defined in the system context (SSAD 2.1).
- Each class described in a constraint, should be described in the implementation Implementation Classes (SSAD 4.1.13).
- IOC: Each connector rule should be implemented by one or more elements described in this connector’s Internal Architecture (SSAD 4.1.8.1.7).

4.1.8.1.7 Internal Architecture

Describe the architecture of this hardware connector classifier that is independent of the implementation technology. Describe the subconnectors of this connector, what they are expected to do, how they are connected, and what they communicate.
4.1.9 Hardware Components

For each hardware component defined in the figure(s) shown in the Hardware Classifier Model (SSAD 4.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware component in the header.

Representation:

The description of hardware components depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

4.1.9.1 Component X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

4.1.9.1.1 Purpose

Describe the purpose of this component.

4.1.9.1.2 Classifier

Identify the classifier of this component.

4.1.9.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a component classifier (SSAD 4.1.5.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this component must satisfy the values specified either by the classifier or this instance.

Representation:

Create a table like Table 18, where a value is specified for each goal that applies to this component’s classifier (SSAD 4.1.5.1.5), where the classifier described as “value is specific to the component instance”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

- Each goal described here, should be described in the L.O.S. Goals of the component’s classifier (SSAD 4.1.5.1.5).
- LCA: If this component is high-risk, then each goal in the L.O.S. Goals of the component’s classifier (SSAD 4.1.5.1.5) that is described as “value is specific to the component instance”, shall have a value.
Guidelines for MBASE System and Software Architecture Description (SSAD)

• IOC: Each goal described in the L.O.S. Goals of the component’s classifier (SSAD 4.1.5.1.5) described as “value is specific to the component instance”, shall have a value.

4.1.10 Hardware Connectors

For each hardware connector defined in the figure(s) shown in the Hardware Classifier Model (SSAD 4.1.2), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the hardware connector in the header.

Representation:

The description of hardware connectors classifiers depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

4.1.10.1 Connector X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill–in the details for the following sections.

4.1.10.1.1 Purpose

Describe the purpose of this connector.

4.1.10.1.2 Classifier

Identify the classifier of this connector.

4.1.10.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a connector classifier (SSAD 4.1.6.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this connector must satisfy the values specified either by the classifier or this instance.

Representation:

Create a table like Table 18, where a value is specified for each goal that applies to this connector’s classifier (SSAD 4.1.6.1.5), where the classifier described as “value is specific to the connector instance”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

• Each goal described here, should be described in the L.O.S. Goals of the connector’s classifier (SSAD 4.1.6.1.5).

• LCA: If this component is high–risk, then each goal in the L.O.S. Goals of the component’s classifier (SSAD 4.1.5.1.5) that is described as “value is specific to the component instance”, shall have a value.

• IOC: Each goal described in the L.O.S. Goals of the connector’s classifier (SSAD 4.1.6.1.5) described as “value is specific to the connector instance”, shall have a value.
4.1.11 Software Components

For each software component defined in the figure(s) shown in the Software Classifier Model (SSAD 4.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the software component in the header.

**Representation:**

The description of software components depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

4.1.11.1 Component X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

4.1.11.1.1 Purpose

Describe the purpose of this component.

4.1.11.1.2 Classifier

Identify the classifier of this component.

4.1.11.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this component. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for a component classifier (SSAD 4.1.7.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this component must satisfy the values specified either by the classifier or this instance.

**Representation:**

Create a table like Table 18, where a value is specified for each goal that applies to this component’s classifier (SSAD 4.1.7.1.5), where the classifier described as “value is specific to the component instance”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Model Integration Rules:**

- Each goal described here, should be described in the L.O.S. Goals of the component’s classifier (SSAD 4.1.7.1.5).

- LCA: If this component is high-risk, then each goal in the L.O.S. Goals of the component’s classifier (SSAD 4.1.5.1.5) that is described as “value is specific to the component instance”, shall have a value.

- IOC: Each goal described in the L.O.S. Goals of the component’s classifier (SSAD 4.1.7.1.5) described as “value is specific to the component instance”, shall have a value.
4.1.12 Software Connectors

For each software connector defined in the figure(s) shown in the Software Classifier Model (SSAD 4.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the software connector in the header.

**Representation:**

The description of software connectors depends on the type of system that you are defining, on the architecture language that you are using to represent your system, and possibly on architecture style considerations.

### 4.1.12.1 Connector X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill-in the details for the following sections.

#### 4.1.12.1.1 Purpose

Describe the purpose of this connector.

#### 4.1.12.1.2 Classifier

Identify the classifier of this connector.

#### 4.1.12.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this connector. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be **measurable, relevant, and specific** (M.R.S.).

The L.O.S. Goals for a connector classifier (SSAD 4.1.8.1.5) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this connector must satisfy the values specified either by the classifier or this instance.

**Representation:**

Create a table like Table 18, where a value is specified for each goal that applies to this connector’s classifier (SSAD 4.1.8.1.5), where the classifier described as “value is specific to the connector instance”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Model Integration Rules:**

- Each goal described here, should be described in the L.O.S. Goals of the connector’s classifier (SSAD 4.1.8.1.5).
- LCA: If this component is high-risk, then each goal in the L.O.S. Goals of the component’s classifier (SSAD 4.1.5.1.5) that is described as “value is specific to the component instance”, shall have a value.
- IOC: Each goal described in the L.O.S. Goals of the connector’s classifier (SSAD 4.1.8.1.5) described as “value is specific to the connector instance”, shall have a value.
4.1.13 Implementation Classes

For each implementation class defined in the figure(s) describing the Internal Architecture (SSAD 4.1.7.1.7) of software components in the Software Classifier Model (SSAD 4.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the implementation class in the header.

Representation:

For each implementation class, describe its purpose, the component that defines it, its interfaces, its operations, its parameters, its state behavior, and its level of services.

Common Pitfalls:

For Simple Systems or Composite Systems, there is a risk of spending too much time describing the behavior (4.1.5.1.4) or the internal architecture (4.1.6.1.7) of a class in detailed.

Model Integration Rules:

- At LCA: Each implementation class that resides in a high-risk component (SSAD 4.1.7.1.7) shall be full defined.
- At IOC: All implementation classes that reside in all software components (SSAD 4.1.7.1.7) in the Software Classifier Model (SSAD 4.1.3), that reside in high-risk components shall be full defined.

577 Guidelines:

Create a UML package, with the stereotype <<class>> and the name “Implementation Class Name Details”, in the package that holds the classifier that represents this software component classifier.

4.1.13.1 Implementation Class X

Based on UML and your chosen implementation language (and, as always, on your risk analysis) fill-in the details for the following sections.

4.1.13.1.1 Purpose

Describe the purpose of this class.

4.1.13.1.2 Defined In: Component Name

Put the full name of the component in which this class resides in the header.

4.1.13.1.3 Interface(s)

Describe the interfaces that are realized by this implementation class.

Representation:

Describe the visible features of the implementation class by creating Create the Static-Structure Diagram named “Interfaces”.

- Add a classifier for the implementation class.
- Add a classifier with the stereotype <<interface>> and the name of the interface for each set of related operations. Define the operations they contained and show any inheritance relations among the interfaces.
Connect the classifier representing the implementation class to each interface with a realization relation.

Note: the class’ Interface(s) (SSAD 4.1.13.1.3) and Parameters (SSAD 4.1.13.1.4) can be represented using one Static–Structure Diagram (name the diagram “Interfaces & Parameters”). The interfaces and parameters descriptions of multiple classes can be shown on the same diagram. (Place the diagram in the package that holds the component’s package.)

For each interface, create a subsection numbered 4.1.13.1.3.x and the name of the interface in the header. The body of the subsection should describe pertinent information about the interface.

Model Integration Rules:

- For each operation interface that a class realizes, the class shall define an operation with the same signature (SSAD 4.1.13.1.7); unless the class is abstract.

- Each interface described here, should be used in the class’s behavior description (SSAD 4.1.13.1.7).

- At LCA:
  - If the class is high–risk, then a set of interfaces for the class shall be described.
  - If the class is not high–risk, then a draft set of interfaces for the class should be described.

- At IOC: the final set of interfaces for the class shall be described.

4.1.13.1.3.1 Interfaces X

Create a section at this level for each interface described above. The header of this section should be the name of the interface and its unique designator, if you have assigned an identifier and it is different from the name. The contents of this section should include a description of the interface’s purpose and a list of all operations defined. For each operation, describe its purpose, its parameters and result (called a signature), and any pre– and post– condition.

4.1.13.1.4 Parameters

Describe any parameters of the implementation class. The parameters need to be set when an instance of the implementation class is created.

Representation:

Create a Static–Structure Diagram that shows the implementation class. Add a dashed rectangular box to upper– right hand corner of the classifier. List the parameters in the box.

Note: the class’ Interface(s) (SSAD 4.1.13.1.3) and Parameters (SSAD 4.1.13.1.4) can be represented using one Static–Structure Diagram (name the diagram “Interfaces & Parameters”). The interfaces and parameters descriptions of multiple classes can be shown on the same diagram. (Place the diagram in the package that holds the component’s package.)

Model Integration Rules:

- At LCA: a preliminary set of parameters for the class, if any exist, should be described.

- At IOC: the final set of parameters for the class shall be described. (If none, say so.)
4.13.1.5 Attributes

Describe the attributes of this implementation class. For each attribute, describe its purpose, its class, its visibility, any default value, and any stereotype.

Representation:

Create a table like Table 21. Describe one attribute of the class in each row of the table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Multiplicity</th>
<th>Ordering</th>
<th>Initial Value</th>
<th>Visibility</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Name</td>
<td>Full name (or name and identifier) of implementation classifier</td>
<td>Optional number of values represented by attribute</td>
<td>ordered</td>
<td>unordered</td>
<td>Expression defining value in class of attribute</td>
<td>Public</td>
</tr>
</tbody>
</table>

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. Give the attribute a name that expresses the mean of its values.
   - A good name is usually a noun (e.g. “color”) or a short noun phrase (e.g. “average duration”).
   - Each name must be unique relative to the owning class. (Two attributes in different classes can have the same name.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   - Clear, self-explanatory names may require several words.

Common Pitfalls:

- Describing attributes that are never used.

Model Integration Rules:

- Each attribute defined in the class should be used by one or more of the classes operations (SSAD 4.1.13.1.6).
- Each class name in an attribute’s type description should be defined in Implementation Classes (SSAD 4.1.13).

4.13.1.6 Operations

Describe the operations of this implementation class. For each operation, describe its purpose, its parameters and result (called a signature), any pre- and post-condition, and its behavior.

Representation:

For each operation, create a subsection numbered 4.13.1.6.x (see example section below) with the name of the operation in the header. Each subsection describe the operations purpose, its parameters and result (called a signature), and any pre- and post-condition, as shown in Figure 6 (you need not repeat the operator name or identifier).
Figure 6: Operation Description

Identifier: Unique identifier for traceability (e.g. Op-xx)

Operation Name: Name of use-case

Parameters: A list of parameter descriptions of the form:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Name</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>Name of parameters</td>
<td>Full name (or name and identifier) of implementation classifier</td>
<td>Expression defining value in class of attribute</td>
</tr>
<tr>
<td>out</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inout</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Result: Full name (or name and identifier) of implementation classifier

Purpose: Brief description of purpose

Pre-conditions: Description of state that the class instance should be in before operation is performed. (informal text, OCL, or both)

Post-conditions: Description the class instance’s state after operation is performed. (informal text, OCL, or both)

Visibility: Public | Protected | Private | Package | (implementation language specific)

Abstract: Yes | No

Method: Overview of the implementation of operation

If the operation is not abstract, then describe the method used to implement the operation; provide a brief text description and explain any critical algorithms; and for non-trivial operation of a class that is not COTS then provide a detailed description as follows:

- If the operation requests operations of other objects, create a Sequence Diagram to illustrate the interactions;
- If the operation performs some complex logic that is not adequately represented by the Sequence Diagram, then create an Activity Diagram to describe the logic.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Recommendations:

1. Give each operation a name that expresses the behavior that it represents.

   - A good name is usually a verb or verb phrase.
   - Each operation in the class must have a unique name and signature. (Two operations in owned by different classes can have the same name and signature.)
   - Avoid names that sound alike or are spelled alike, as well as synonyms.
   - Clear, self-explanatory names may require several words.
Common Pitfalls:

- Describing operations that are never used. Each operation should be requested in at least one description of either use-case realization of its containing component (SSAD 4.1.8.1.7) or a method of this class or another class (SSAD 4.1.13.1.6).
- Not describing all operations that are used in either use-case realization or a method of this class or another class.

Model Integration Rules:

- Each operation defined in the class should be requested in at least one description of either use-case realization of its containing component (SSAD 4.1.8.1.7) or a method of this class or another class (SSAD 4.1.13.1.6).
- Each class name in a parameter’s description or operation result should be defined in Implementation Classes (SSAD 4.1.13).
- At LCA:
  - The operations for the class that are used in high-risk behaviors shall be fully described, including any Sequence or Activity Diagrams.
- At IOC:
  - The operations for the class that are used in the build shall be fully described, including any Sequence or Activity Diagrams.
  - Draft descriptions of other operations should be provided.

4.1.13.1.6.1 Operation X

Create a section at this level for each operation of the class. The header of this section should be the name of the operation and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 4.

4.1.13.1.7 State Behavior

If significant, describe the state behavior of the class, i.e. how the behavior of a class instance depends on the state it is in. Describe the states of the class instance, the events that cause the class instance to change states, and how the processing is different in each state.

Representation:

Create a State Model that describes the class instance’s state behavior. Represent the model as a package with the stereotype <<State Model>> with the package name “States of Implementation Class Name”. Create one or more UML Statecharts that show the states of class instances, the events (e.g. receipt of operation requests) that cause state changes, and the actions performed by the instance (e.g. request another object perform one of its operations) either when in a state or when changing states.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).
Guidelines for MBASE System and Software Architecture Description (SSAD)

577 Guidelines:

In Rational Rose, instead of creating a separate package for the State Model, right-click on the class in the Browser View and select New | Statechart Diagram. Rose will create a model named “State/Activity Modelx” that is attached to the class, and a diagram named “NewDiagram” in the model. Rename the State Model to “State behavior” and the diagram to “Top Level”.

Recommendations:

1. Describe the state behavior of the class if

   • The class participates in component/system processes that have high safety or high reliability L.O.S. requirements;

   • An instances of the class represents a concurrent element (e.g. a OS process or thread) or represents a resource can be shared by two or more concurrent elements;

   • Some of the class’ operations can only be executed when an instance is in a certain state or the details of some methods change depending on the state of the instance.

2. For additional recommendations, see Recommendations for system’s Modes of Operation (SSAD 2.3.2).

Common Pitfalls:

• Spending too much time describing the states in detailed for low–risk classes or classes that do not have significant modal behavior. This problem is more common for systems that do not have high safety or reliability L.O.S. requirements.

• For additional pitfalls, see Common Pitfalls for system’s Modes of Operation (SSAD 2.3.2).

Model Integration Rules:

• Each operation of this class (SSAD 4.1.13.1.6) should appear in the class’ state model as an event for one or more transitions (internal or between states).

• Every operation requested by an instance of this class in any Sequence Diagram created to describe the implementation of a component’s process (SSAD 4.1.8.1.7) or the method of another operation (SSAD 4.1.13.1.6).

• Each operation described as an event for one or more transitions (internal or between states) shall be described in this class’ operations (SSAD 4.1.13.1.6).

• Each operation requested in an action or condition shall be described in the operations of the class of the instance to which the request is sent (SSAD 4.1.13.1.6).

• At LCA: a draft behavior for the class shall be described if the class is involved in high–risk behavior.

• At IOC: the behavior for the class shall be described if the class is included in the build.

4.1.13.1.8 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this implementation class. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).
Guidelines for MBASE System and Software Architecture Description (SSAD)

The L.O.S. Goals for an implementation class defines the goals for all instances of this classifier, and the goals that an implementation of this implementation class must achieve. The classifier’s goals are used to demonstrate that the architecture will meet the containing component’s goals (i.e. the component’s goals will be meet its goals if all classes achieve their goals).

**Representation:**

For each system goal that applies to this implementation class fill out the L.O.S. Form (Table 11). Only specify value(s) in the Measurable field only if the value(s) specified apply to all instances. If each instance can have its on value, then specify that the “value is specific to the class instance”. Optionally specify a default value or a limiting value.

**Recommendations:**

See Recommendations for L.O.S. Goals of architecture–design Software Component Classifiers (SSAD 3.1.7.1.5).

**Common Pitfalls:**

- Not satisfying the M.R.S. criteria.
- Specifying values that do not apply to all instances.
- Specifying values for a goal that exceeds values, or is outside the range of values, specified for the architecture–design class that this class implements.

**Model Integration Rules:**

- Each class L.O.S. Goals (OCD 4.4) should be described here.
- Each operation described in a goal, should be described in an implementation class’s Operations (SSAD 4.1.13.1.6).
- Each attribute described in a goal, should be described in an implementation class’s Attributes (SSAD 4.1.13.1.5).
- Each state described in a goal, should be described in the implementation class’s State Behavior (SSAD 4.1.13.1.7).

**4.1.13.1.9 Constraints**

Describe any constraints on the use and implementation of this class that are not captured in other sections of the class description. The following paragraphs describe some typical constraints.

- Rules that the class must implement in support of rules that the system must satisfy;
- Constraints imposed on class’s implementation by the architecture style(s), patterns, or frameworks used for system architecture;
- Constraints imposed on class’s implementation by architecture design notation used for system architecture. (These are less likely.)

The goal is to document accurately important rules that affected the class’s design and implementation, without getting into implementation details.

**Representation:**

See Representation for Constraints of architecture–design Software Component Classifiers (SSAD 3.1.7.1.6).
Trade–offs:

See Trade–offs in section OCD 3.5(current organization rules).

Recommendations:

1. To facilitate traceability, assign each rule a unique number (e.g. Class-Rule-01 or CR-01).

2. For additional recommendations, see Recommendations in section OCD 3.5(current organization rules).

Common Pitfalls:

- Describing a rule that is already captured in another view of the class (e.g. state behavior of the class (SSAD 4.1.13.1.7)).

Model Integration Rules:

- Each system rule described here should be described in the proposed system’s rules (SSAD 2.5).
- Each operation described in a constraint, should be described in an implementation class’s Operations (SSAD 4.1.13.1.6).
- Each attribute described in a constraint, should be described in an implementation class’s Attributes (SSAD 4.1.13.1.5).
- Each state described in a constraint, should be described in the implementation class’s State Behavior (SSAD 4.1.13.1.7).
- Each class described in a constraint, should be described in the architecture’s Implementation Classes (SSAD 4.1.13).
- IOC: Each class rule should be implemented by one or more elements described in the implementation (e.g. the code).

4.1.14 Objects

For each object defined in the figure(s) shown in the Software Classifier Model (SSAD 4.1.3), create a subsection with the name and unique identifier (if you have assigned an identifier and it is different from the name) of the object in the header.

Representation:

For each object, describe its purpose, its classifier, and any L.O.S. goals.

4.1.14.1 Object X

Based on your chosen architectural style and language (and, as always, on your risk analysis) fill–in the details for the following sections.

4.1.14.1.1 Purpose

Describe the purpose of this object.
4.1.14.1.2 Classifier

Identify the classifier of this object.

4.1.14.1.3 L.O.S.

Describe the system L.O.S. requirements (SSRD 5) and goals (SSAD 2.4) that apply to this object. (Some system L.O.S. requirements may not apply.) Each L.O.S. requirement should be measurable, relevant, and specific (M.R.S.).

The L.O.S. Goals for an object’s implementation class (SSAD 4.1.13) defines which system goals apply to all instances of this classifier and values that apply to all instances. In this section, define values for those goals that the classifier described. The implementation of this object must satisfy the values specified either by the classifier or this instance.

Representation:

Create a table like Table 18, where a value is specified for each goal that applies to this an object’s implementation class (SSAD 4.1.13), where the classifier described as “value is specific to the object”.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

Model Integration Rules:

- Each goal described here, should be described in the L.O.S. of the object’s implementation class (SSAD 4.1.13).
- LCA: If this object is high–risk, then each goal in the L.O.S. of the object’s implementation class (SSAD 4.1.13) that is described as “value is specific to the object”, shall have a value.
- IOC: Each goal described in the L.O.S. of the object’s implementation class (SSAD 4.1.13) described as “value is specific to the object”, shall have a value.

4.2 Behavior

Describe how the components work with each other and with the actors to implement the required behavior of the system (SSAD 2.3).

Describe how each system process (SSAD 2.3) is implemented by the components described in the Deployment Model (SSAD 4.1.4). For each process, identify which components participate in the system process, and which instances of the classes (SSAD 4.1.13), which are implementations of Analysis Classes (SSAD 3.2), are inspected, manipulated, or produced in the process; and describe the interactions among the components and the classes that are implementations of analysis classes.

Representation:

See Representation in the Behavior section (SSAD 3.3) of architecture design.

For each process implementation, create a subsection numbered 4.2.x (see example section below) with the name of the process implementation in the header. Each subsection should provide the information shown in the Figure 5 (on page 6) and one or more Interaction Diagrams.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).
Recommendations:

1. Start an Interaction Diagram by creating one or more instances of each actor that participates in the use-case realized. List the actions specified in the courses of action for the use-case realized along the left margin of the diagram. For each action,
   - Identify component(s) of the system that should participate and instance(s) of implementation classes that are needed. (The activity diagram of the use-case should be helpful in identifying the analysis classes are involved.)
   - Identify process(s) that each component should do. (Each process should be described in the behavior of the component’s class (SSAD 4.1.7.1.4.1).)
   - Identify operation(s) on the instances of implementation classes need to be performed. (Each operation should be described in the class’ description (SSAD 4.1.13).)
   - Show message(s) that need to be exchanged.

2. For additional recommendations, see Recommendations in the Behavior section (SSAD 3.3) of architecture design.

Common Pitfalls:

See Common Pitfalls in the Behavior section (SSAD 3.3) of architecture design.

Model Integration Rules:

- For LCA,
  - Each system process (SSAD 2.3.1) that is high-risk, architecturally-significant, or particularly complex should have at least one implementation.
    - A draft Use-Case Realization Description should be filled out.
    - A draft Implementation Diagram should be created.
      - Each actor instance in the diagram should be an instance of an actor associated with the system process (SSAD 2.3.1).
      - Each non-actor instance in the diagram should be an instance of either a software component, or interface in the architecture (SSAD 3.1.11), or the implementation class (SSAD 4.1.13).
      - Each message in the diagram should be the name of a process, an operation, or an event defined for either a software component or interface in the architecture (SSAD 3.1.11), or an implementation class (SSAD 4.1.13).
      - Each action described in courses of action for the use-case realized by one or more messages.

- For IOC,
  - Each system process (SSAD 2.3.1) that is high-risk, architecturally-significant, or particularly complex should have at least one implementation.
    - Its Use-Case Realization Description shall be filled out.
- An Implementation Diagram shall be created.
  - Each actor instance in the diagram shall be an instance of an actor associated with the system process (SSAD 2.3.1).
  - Each non-actor instance in the diagram shall be an instance of either a software component, or interface in the architecture (SSAD 3.1.11), or the implementation class (SSAD 4.1.13).
  - Each message in the diagram shall be the name of a process, an operation, or an event defined for either a software component or interface in the architecture (SSAD 3.1.11), or an implementation class (SSAD 4.1.13).
  - Each action described in courses of action for the use-case realized by one or more messages.
  - For IOC, each system process (SSAD 2.3.1) shall have at least one implementation described to the same level of detail as a high-risk, architecturally-significant, or particularly complex implementation at LCO.

### 4.2.1 Process Realization X

Create a section at this level for each process implementation. The header of this section should be the name of the process implementation and its unique designator, if you have assigned an identifier and it is different from the name.

Provide the information described in Figure 5. (If the information duplicates the information for a process implementation described during architecture analysis (SSAD 3.3).

### 4.3 L.O.S.

Describe the elements of the architecture (components, connectors, analysis classes, etc.) to which each system L.O.S. applies and the projected value of each system L.O.S.

The goal is to evaluate the L.O.S. that the architecture will provided (assuming each component, connector, and class achieves its goals). This activity provides technical details supporting the FRD.

**Representation:**

See Representation in the L.O.S., Projected section (SSAD 3.4) of architecture design.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

See Recommendations in the L.O.S., Projected section (SSAD 3.4) of architecture design.

**Common Pitfalls:**

See Common Pitfalls in the L.O.S., Projected section (SSAD 3.4) of architecture design.

**Model Integration Rules:**

See Model Integration Rules in the L.O.S., Projected section (SSAD 3.4) of architecture design.
4.4 Patterns & Frameworks

Describe any implementation architecture styles (e.g. the Prism style [http://sunset.usc.edu/~softarch/Prism/]), patterns (e.g., pipe–and–filter and client–server), or frameworks (e.g. Java and CORBA) used to describe the system architecture.

**Representation:**

See Representation in the Architectural Styles, Patterns & Frameworks section (SSAD 3.5) of architecture design. Include descriptions of features of the style, pattern, or framework description.

Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD 5).

**Recommendations:**

See Recommendations in the Architectural Styles, Patterns & Frameworks section (SSAD 3.5) of architecture design.

**Model Integration Rules:**

- Each style, pattern, or framework should be used to define the implementation structure (SSAD 4.1).

- Each benefit and cost described here, should be related to the system’s L.O.S. Goals (SSAD 2.4) and should be factored in to the projected L.O.S. for the architecture (SSAD 4.3).

- The details of each limitation should be captured in the implementation structure (SSAD 4.1).

**UML Guideline:**

Create a package name “Frameworks”. For each framework (e.g. Java), create a package with the name of the framework. In each framework package, create Static–Structure Diagrams that describe any classes, interfaces, or objects that are defined in the framework and are used by the implementation.

**577 Guideline:**

Follow UML Guideline.

4.5 Project Artifacts

Describe how the software components and classes defined in the implementation Structure (SSAD 4.1) are assigned into project artifacts (e.g. files, database tables, web pages) that will be used to produce code; how the files are organized into directories; the dependency relations among the artifacts and directory structure.

**Representation:**

Create a hierarchical list of directories, their files, and the classes or components in each file. If all the classes of a component are in the same file, then just list the component name; otherwise list the specific classes. For example:

```
Directory Name 1
  File Name 1
    Component Classifier Name 1
  File Name 2
    Component Classifier Name 2
```
Add any terms identified during this modeling effort that are unique to the organization or have unique meaning to the Glossary for System Analysis and Design (SSAD).

**Recommendations:**

1. Include all required files, scripts, programs, images, and libraries in the directory structure.

2. Do not describe details of COTS products.

3. One strategy for defining the directory structure is to parallel the package hierarchy in the Software Classifier Model (SSAD 4.1.3). Another strategy is to create packages that reflect different categories of artifacts. Mixed strategies are common.

**Model Integration Rules:**

- At LCA, the Configuration Model should show the configuration of any high-risk components and classes.

- At IOC, the Configuration Model shall show the configuration of all components and classes.

**UML Guideline:**

Create a package with the label “Configuration Model”. In this package, create one or more UML Component Diagrams that show the project artifacts; the project directories; the contents of each project artifact; and the dependency relations among project artifacts and directories. Represent each project artifact as a classifier icon with one of the UML stereotypes shown in Table 22, or with an implementation- or platform-specific stereotype (e.g. Webpage, jarFile, script).
Table 22: Stereotypes for Project Artifacts

<table>
<thead>
<tr>
<th>Stereotypes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;file&gt;&gt;</td>
<td>a physical file that is otherwise undifferentiated</td>
</tr>
<tr>
<td>&lt;&lt;table&gt;&gt;</td>
<td>a database table</td>
</tr>
<tr>
<td>&lt;&lt;document&gt;&gt;</td>
<td>a generic file that is not a «source» file or «executable»</td>
</tr>
<tr>
<td>&lt;&lt;executable&gt;&gt;</td>
<td>a file that can be executed on a computer</td>
</tr>
<tr>
<td>&lt;&lt;library&gt;&gt;</td>
<td>a static or dynamic library file.</td>
</tr>
<tr>
<td>&lt;&lt;source&gt;&gt;</td>
<td>a compilable source code file</td>
</tr>
<tr>
<td>&lt;&lt;document&gt;&gt;</td>
<td>a generic file that is not a source or executable file</td>
</tr>
</tbody>
</table>

Create a Component Diagram that shows the high–level directories. Represent each directory used to organize the project artifacts as a package. In each package, create a Component Diagram that shows the contents of the directory represented by the package.

If the contents of one package depend on the contents of another package, add a dependency relation from the client package to the provider. If a project artifact depends on another project artifact or the contents of a package, add a dependency relation from the client to the provider.

For each artifact, create a UML Component diagram in the package containing the project artifact that shows the project artifact and any components or implementation classes that are in the project artifact. If a project artifact contains one or more components, then either nest each component in the project artifact, or show an aggregation relation from the artifact to each component in the artifact. If an artifact contains only a subset of the classes of a component, then either nest each class in the project artifact, or show an aggregation relation from the artifact to each implementation class in the artifact.16

Multiple artifacts and their contents can be shown on one diagram, if the diagram is readable and clear. For example, create a diagram that shows all project artifacts that are webpages, and another that shows all project artifacts that are database files. It is often necessary to include one or more artifacts from another package on a diagram that is otherwise devoted to a particular package in order to show dependencies on those artifacts.

Rational Rose Guideline:

Rose’s Component Diagram only allows dependency relations between diagram elements, and does not allow classifiers on the diagram. Either:

1. Follow the directions given in the UML Guidelines, except create Static–Structure Diagrams instead of the Component Diagrams, and represent each project artifact as a classifier with one of the UML stereotypes shown in Table 22, or with an implementation– or platform–specific stereotype (e.g. Webpage, jarFile, script).

2. Follow the directions given in the UML Guidelines, except

   • Instead of creating package with the label “Configuration Model”, use the predefined Rose package labeled “Component View”;

16 In UML v1.4, there is a conflict between the description of the relations to use in the Semantic Section and the description in the Notation Section of the manual. We have used the aggregation relation as described in the Semantic Section for the relation between project artifacts and both components and classifiers.
Guidelines for MBASE System and Software Architecture Description (SSAD)

- Create Component Diagrams that show the packages and project artifacts (do not forget the stereotypes);

- Open the Specification for the project artifact, switch to the “Realizes” tab, and select the classifiers that are contained in the project artifact. (A potentially faster way to assign a classifier to a component is to drag the classifier from the browser view and “dropping” it on the project artifact on a Component Diagram.)

The first approach results in a model that closely resembles the UML Specification. The second approach, using Rose’s Deployment View, has the advantage that ROSE’s code generator will use the information in the Component View when generating code.

577 Guideline:

Follow ROSE Guidelines for using Rose’s Component View (#2).

5. Glossary for System Analysis and Design

Create an alphabetical listing of all uncommon or organization-specific terms, acronyms, abbreviations, and their meanings and definitions, to understand the Domain Description.

Recommendation:

Glossary items are often answers to questions that you ask to the client: “What does this mean?”

6. Appendices

Create appendices as need. Each appendix shall be referenced in the main body of the document where the data would normally have been provided.

Include supporting documentation or pointers to electronic files containing:

- Policies (e.g., applicable Copyright Laws)
- Descriptions of capabilities of similar systems
- Additional background information
Life Cycle Plan (LCP)

A. Description

1. Purpose

- To serve as a basis for monitoring and controlling the project’s progress
- To provide general information during Inception and Elaboration about those project management areas which may get their own documents for Construction
- To serve as the basis for controlling the project's progress in achieving the software product objectives
- To help make the best use of people and resources throughout the life cycle
- To provide evidence that the developers have thought through the major life cycle issues in advance
- Organized to answer the most common questions about a project or activity: why, what, when, who, where, how, how much, and whereas

2. Completion Criteria

Below are the completion criteria for the Life Cycle Plan for the three phases:

- Life Cycle Objectives (Inception Phase)
- Life Cycle Architecture (Elaboration Phase)
- Initial Operational Capability (Construction Phase)

2.1 Life Cycle Objectives (LCO)

- Identification of life-cycle stakeholders
  - Users, customers, developers, maintainers, interfacers, general public, others
  - Identification of life-cycle process model
- Top-level phases, increments
- Major risks identified
- Deliverables, budgets and schedules achievable by at least one system/software architecture
- Quality Management history (summary) and plans for Elaboration phase

2.2 Life Cycle Architecture (LCA)

- Elaboration of WWWWWWHH for Initial Operational Capability (IOC)
• All major risks resolved or covered by risk management plan
• Deliverables, budgets and schedules achievable by the architecture in the SSAD
• Quality Management history summary and plans for Construction phase

2.3 Initial Operational Capability (IOC)
• Plans for future increments beyond IOC

3. Intended Audience
Primarily the performer teams in each stage, but also important for customers, and useful for other stakeholders.

4. Participating Agent
The Project Manager for each stage leads the baselining of the plan for that stage. Plans for future stages are normally developed by a designated team member during Engineering Stage. Stakeholders affected by plan elements should participate in their definition.

5. Performing Agent
Development team

6. High-Level Dependencies
Products specified by Requirements and Architecture must be buildable and supportable within the budgets and schedules in the Life Cycle Plan. Plans for transition and support must be consistent with the Operational Concept Description.

7. Overall Tool Support
Use of planning and control tools such as Microsoft Project is advised. A combination of estimation models such as COCOMO II and performer-determined task completion estimates should be used to help develop budget and schedule estimates. If COCOMO II estimates conflict with good engineering judgment, use and document the engineering judgment. Effort data collection procedures should be used to collect progress metrics and analyze project effort.

Potential Pitfalls/Best Practices: It should be noted that through the high degree of dependencies between tasks and people, delays in critical areas might cause schedule problems in many (if not all) activities later on. Even the simplest reasons such as a vacation of a key person not considered may be responsible for such schedule upsets. Utmost care should therefore be devoted into planning and maintaining this document, and to ensuring its feasibility via the Feasibility Rationale Description and subsequent progress monitoring and control activities.

Quality Criteria: The life cycle plan adds a time component to the other documents. Thus, there is a high degree of dependency especially between the SSRD (and SSAD) and the LCP. If the tasks identified in this document do not reflect the requirements and the components of the product to be developed, then the plan will be useless. Thus, maintaining the conceptual integrity between the tasks and the things they create/solve is a prime quality criterion.

Further, the timing and scheduling of the tasks is highly dependent on not only the SSAD and SSRD as explained above, but also on the people who are working on them. Fortunately, that part is fairly independent from other documents. However, the ultimate core of this document is the creation of a responsibility trace (or matrix) which
maps the people to components via tasks. The plan is a record of the personal and organizational commitments of each of the stakeholders to their part of the overall project. If these commitments are vaguely defined or poorly matched to people's capabilities, there is a high risk of project misunderstandings, frustrations and failures. The bottom-line quality criteria are the assurance of stakeholder commitment, resource level feasibility and business-case return on investment in the Feasibility Rationale Description. The life cycle plan is also a summary record of the personal and organizational performance with respect to Quality Management related activities.

8. Degree of Detail and Tailoring

The degree of details of the LCP should be risk-driven (as with any MBASE model). If it’s risky to put an item in (e.g., detailed schedules or work breakdown staff assignments), don’t put it in. If it’s risky not to put an item in (e.g., critical stakeholder review milestones, pre-scheduled non-negotiable events), do put it in. Sections of the LCP may be tailored down or consolidated for small or non-critical, well defined systems.

B. Document Sections

1. Introduction

1.1 Purpose of the Life Cycle Plan Document

- Summarize the purpose and contents of this document with respect to the particular project and people involved
- Avoid generic introductions as much as possible: instead, show how your particular Life Cycle Plan meets its completion criteria for the given phase

Common Pitfalls:

- Simply repeating the purpose of the document from the guidelines

1.2 Assumptions

This section identifies the conditions that must be maintained in order to implement the plans below within the resources specified. If one or more of these assumptions is no longer satisfied, then the requirements (System Requirements or Project Requirements) may need to be re-negotiated; or the Life Cycle Plan may need to be re-evaluated.

Develop a list of assumptions on which the project planning decisions are based so that everyone on the project understands them. It is important to uncover unconscious assumptions and state all assumptions up front. Assess and state the likelihood that the assumption is correct, and where relevant, a list of alternatives if something that is assumed does not happen.

These assumptions might cover such items as:

- Stability of software product requirements, including external interfaces
- Stability of required development schedules
- Continuity of funding
- On-schedule provision of customer-furnished facilities, equipment, and services
- On-schedule, definitive customer response to review issues and proposed changes
• What the developers expect to be ready in time for them to use. For example, other parts of your products, the completion of other projects, software tools, software components, test data, etc.

• Dependencies on computer systems or people external to this project

• Any tasks which are on the critical path of the project

• [Consistent with Process Match to System Priorities (FRD 3.2)]

• [Consistent with Project Requirements (SSRD 2)]

• [Consistent with Assumptions and Results Chain (OCD 2.1)]

Integration and Dependencies with other components:

Assumptions are made because the necessary detail is not yet known. Thus, this section will change as the system evolves. The assumptions can be based on any aspect of the development and thus they can be dependent on any document (e.g. OCD, SSRD, and SSAD). The risk analysis part (LCP 4.1.4) should reflect those assumptions.

1.3 References

• Provide complete citations to prior and current related work and artifacts, documents, meetings and external tools referenced or used in the preparation of this document

• Useful for consistency checking and traceability

577 Guidelines:

A "complete citation" for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]

1.4 Change Summary

577 Guidelines:

For versions of the LCP since the last ARB, include a summary of changes made in the document to ease the review process.

2. Milestones and Products

This section tells what project functions will be performed during life cycle, and what products will be developed. It contains schedules and milestones that indicate when each function and product will be completed.

Integration and Dependencies with other components:

• The milestones defined for stages and phases in LCP 2 must be consistent with the stage and phase responsibilities, reviews, budgets, etc. in LCP 3, 4, and 4.4.1.

• The content of the LCO, LCA, and RLCA (re–baselined) milestones is exactly the content of the OCD, SSRD, SSAD, Prototype, LCP, and FRD for those milestones.
2.1 Overall Life Cycle Strategy

Describe the overall approach to be used in defining, developing, and supporting the product:

- The choice of process model(s) used (waterfall, evolutionary, spiral, etc.) and the major life cycle stages, phases and milestones. Departures from this approach can be identified in part (d).
  
  a. If prototyping is employed, the nature and phasing of the planned prototypes.
  
  b. If incremental development is employed, describe the content and phasing of the successive increments to be developed. The phasing of the increments should correspond to the system priorities.
  
  c. Specifics of any other departures from the approach in item (a) (e.g., design-to-cost, design-to-schedule, multiple parallel design or development teams, product-line as well as product development).
  
  d. Top-level milestones (e.g. Gantt charts) and activity dependencies (e.g. PERT charts) showing the sequence of major products and activities.

Focus on the external products and milestones that the customer will see. Later sections should give the internal project details. Emphasize the critical process drivers and process strategies analyzed in FRD 3.2.

The process drivers will likely include the production stage, including the need to fully transition the product to the client. Additional process drivers could include client infrastructure constraints, dependencies on other projects, and challenging performance or dependability requirements.

**Common Pitfalls:**

- Only identifying the current stage in the life cycle strategy
- Confusing the stages and phases of the life cycle
- Not identifying the need for transition and support

**577 Guidelines:**

The recommended process model is the Win–Win Spiral Model; and the major life-cycle stages, phases, and milestones are (See Figure 1):

- **Engineering Stage**
  
  - Inception Phase (Life Cycle Objectives): one Win–Win Spiral cycle, completed by an LCO ARB review
  
  - Elaboration Phase (Life Cycle Architecture): one Win–Win Spiral cycle, completed by an LCA ARB review

- **Production Stage**
  
  - Construction Phase: a short Win–Win Spiral cycle, to incorporate changes that may have occurred since the LCA milestone, and to incorporate Product Improvement suggestions from the Individual Critiques, completed by an Architecture Rebaseline Review; an initial core-capability development increment; followed by several risk-driven incremental iterations or design-to-schedule adjustments, completed by a Transition Readiness Review
  
  - Transition Phase, completed by a Product Readiness Review

- **Support Stage** (USC-ISD responsibility)
• A series of releases, each with its appropriate choice of the stages and phases above, completed by a
  Product Readiness Review

The process drivers will likely include the production stage, including the need to fully transition the product to the
client. Additional process drivers could include client infrastructure constraints, dependencies on other projects, and
challenging performance or dependability requirements.

2.2 Phase Milestones and Schedules

Provide more detailed milestones (e.g. Gantt charts) and activity dependencies (e.g. PERT charts) identifying the
activities (in each increment, if applicable), showing initiation and completion of each activity and depicting
sequential relationships and dependencies among activities. Identify those activities that impose the greatest time
restrictions on the project (i.e., which are on the critical path). The activities described here should be tracked with
progress metrics. Artifacts to be delivered at the end of each phase should be identified and the completion criteria
for each artifact should be specified. For each artifact (plans, specifications, manuals, reports, code, data,
equipment, facilities, training material, person-hours, etc.) to be delivered to the customer, provide the following
information on the nature of the deliverable:

1. The name or title of the artifact

2. The date on which the artifact is due

3. The required format of the artifact (disk format, tape format, document format, etc.)

4. The completion or "exit" criteria for each artifact (evidence required of being produced, delivered,
   received, tested, approved, etc.)

5. Pointers to relevant contract requirements

Completion criteria are defined in [Royce, 1998] Chapter 5 and Section 9.1. The criteria will vary by phase but
often involve stakeholder concurrence, evidence of completeness and stability, actuals vs. planned estimates,
prototype acceptability, and more. The completion criteria for each of the components of the LCO and LCA
package components are indicated in this document.

Common Pitfalls:

• Not providing a graphical representation of the entire life cycle

• Only describing the current stage in the life cycle

• Not identifying the deliverables at each milestone (both minor and major)

• MBASE Win–Win Spiral increments are different than Win–Win Spiral development increments. The MBASE
  Win–Win Spiral is in invariant aspect of using MBASE whereas the general Win–Win spiral is a lifecycle
  process choice that must be adapted to each lifecycle phase it is to be used in. This typically consists of
  specifying the type and duration (entry and exit conditions) of each spiral increment and the number of times
  the increments are visited. Avoid confusing the two processes, however the MBASE Win–Win spiral is a
  fundamental driver for any lifecycle process, including when other Win–Win Spiral’s are used within
  development phases.

Elaborate on the top-level information given in LCP 2.1. The following example illustrates the typical level of
detail to be provided for software integration and test activities:
• Section 2.1. For each increment, indicate completion of integration, of item qualification test(s), and of system qualification test; and indicate major dependencies on life cycle activities, on other increments, on facilities, etc.;

• Section 2.2. Indicate milestones showing the overall order in which components are integrated, and the intermediate stages of increment and system qualification testing. Show how these are synchronized with milestones for test preparation (drivers, facilities, equipment, data, post-processors, etc.) for the various increments.

• Detailed Integration and Test Plan (not part of LCP). Indicate the integration order-of-build for all software components. Identify each test to be performed and indicate which itemized requirement(s) it will test, and where it fits in the overall sequence of tests.

• [Consistent with Process Match to System Priorities (FRD 3.2)]

577 Guidelines:
The recommended process model is the Win–Win Spiral Model; and the major life-cycle stages, phases, and milestones are (See Figure 1):

• Engineering Stage
  
  • Inception Phase (Life Cycle Objectives): one Win–Win Spiral cycle, completed by an LCO ARB review
  
  • Elaboration Phase (Life Cycle Architecture): two Win–Win Spiral cycles, each completed by an LCA ARB review. The second, called the re-baselined LCA, is a short Win–Win Spiral cycle, to incorporate changes that may have occurred since the LCA milestone, and to incorporate Product Improvement suggestions from the Individual Critiques.

• Production Stage
  
  • Construction Phase: two cycles – an initial core-capability development increment; followed by several risk-driven incremental iterations or design-to-schedule adjustments, completed by a Transition Readiness Review
  
  • Transition Phase, completed by a Product Readiness Review

• Support Stage (USC-ISD responsibility)
  
  • A series of releases, each with its appropriate choice of the stages and phases above, completed by a Product Readiness Review

The process drivers will likely include the production stage, including the need to fully transition the product to the client. Additional process drivers could include client infrastructure constraints, dependencies on other projects, and challenging performance or dependability requirements.

2.2.1 Engineering Stage (Inception and Elaboration Phases)

Identify the specific activities, milestones, and their priorities for the inception and elaboration of the system within a feasible (with respect to current and anticipated constraints such as schedule, budget, and personal) and timely manner to satisfy LCO and LCA exit criteria. Be sure to account for document creation (e.g. OCD, SSRD, SSAD, LCP, FRD), review preparation, meetings, Win–Win negotiations, research, and other time intensive activities.
2.2.2 Production Stage (Construction and Transition Phases)

Identify the specific activities, milestones, and their priorities for implementation of the system requirements in a feasible timely manner to achieve initial operational capability (IOC). Mainly this should focus on the schedule for which the system components will be implemented (i.e., coding activities). The activities should account for the execution (and perhaps further development) of IOC plans (iterations, quality, test, peer review, transition, etc.) and the creation of major client deliverables such as user manuals; test plan(s), test description and results, peer review and iteration reports; release notes, final architecture documents). Be sure to account for reviews and inspections (including preparation for them), LCA and IOC documentation updates, training, component configuration, and so forth. Risks often effect the schedule in critical ways during production hence it is essential to account for possible variations due to risk exposure and mitigations.

2.2.3 Support Stage

Identify and schedule the necessary or desired system support activities for when the system is in operation (post system transition). Indicate the degree of client concurrence on any client commitments expressed or implied for the support of the system post-transition. It is critical to account for maintenance and evolutionary activities to ensure the long term viability of the system. In particular be mindful of the organization's growth vectors and future likely technical/cultural shifts. For example if it is known that the organization will be adopting a new database within a given time frame post system transition that the system will be required to use, then some support activity should address how the system will be transitioned to the new database. Many of the activities may involve stakeholders outside the development team, perhaps not participating in the engineering or production stages. It is essential that such stakeholders are identified in advance (if possible) and are part of the overall Win–Win agreements for the operation and support of the system.

Common Pitfalls:

Not identifying and explicitly scheduling significant effort tasks such as ARB review preparation, peer reviews, document creation, research, configuration, etc.

2.3 Project Deliverables

2.3.1 Engineering Stage

Engineering Stage deliverables include the LCO/LCA versions of the:

- Operational Concept Description
- System and Software Requirements Description
- System and Software Architecture Description
- Life Cycle Plan
- Feasibility Rationale Description
- Prototype and optional Prototyping Results (See OCD 5.)

Draft versions of the Construction Phase "Plans", with content determined on a risk driven basis, of the:

- Iteration Plans
- Quality Management Plan
• Test Plan(s)
• Peer Review Plan

2.3.2 Production Stage

Internal deliverables include the following documents and the added transition package to be delivered to the client:

• LCA package (kept up-to-date with "as-built" architecture and design)
• Iteration Plans
• Iteration Assessment Reports
• Release Description (for each internal release)
• Quality Management Plan
• Test Plan(s)
• Test Description and Results
• Peer Review Plan
• Peer Review Reports

The transition package contents includes the software library:

• Source Code
• Load Modules
• Installation Scripts
• Software Development Files (e.g. test modules)

Client-side deliverables include:

• User Manual
• Training Plan
• Transition Plan
• Support Plan
• Training materials (including tutorials and sample data)
• Regression Test Package
• Tools and Procedures
• Version Description Document (or Final Release Notes)
• Facilities, Equipment and Data (these may not be the responsibility of the team)
Guidelines for MBASE Life Cycle Plan (LCP)

577 Guidelines:

- Use actual milestone dates rather than week numbers.
- Develop phase capabilities concurrently rather than sequentially.
- Transition planning and preparation should be done to allow for an initial delivery to the customer two weeks before the end of the semester, followed by 2 weeks of installation, hands-on training, usage testing, refinement, and completion of IOC deliverables.
- Provide graphical activity charts: in particular, show externally provided resources and components, and highlight the ones (e.g., equipment purchase) which are on the critical path for the transition.
- The following figure illustrates the overall CS577 milestones throughout the project lifecycle:

![Figure 7: CS577 Lifecycle Milestones](image)

3. Responsibilities

This section tells who will be responsible for performing the various software life cycle functions, and where organizationally they will be performed. It identifies the major life cycle-related agents (developer, customer, maintainer, users and interfacers) and establishes their roles and responsibilities. It defines the major organizational components of the project, and indicates their responsibilities across the phases of the life cycle.

It presents organization charts showing individual and organizational responsibilities, and includes plans for project staffing and training.

- [Consistent with OCD 4.2.7]

Integration and Dependencies with other components:

This section is relatively independent from other documents. However, within the SDP this section describes human resources, which are needed in other sections of this document. The availability of people is the basis for scheduling.
and creating milestones in LCP 2. because the people identified in this section must be connected to the tasks which were identified in LCP 2. Together, the responsibilities can be defined and possible inconsistencies can be eliminated. Inconsistencies are for example:

- no one assigned to a task
- too many assigned to a task (or it is not clear who is primarily responsible for it)
- input to a task may not be available, etc.

It is particularly important to ensure that the non-developer stakeholders (users, operators, customers, etc.) concur with any responsibilities assigned to them in the LCP.

### 3.1 Stakeholder Responsibilities

Identify which organizations will assume responsibility for carrying out the functions of the major life cycle-related agents: developer, customer, maintainer, users, and interfacerers. Adapt Table 23 as appropriate to indicate the life-cycle responsibilities of each agent. Indicate the roles and responsibilities of subcontractors, vendors, team members, independent verification and validation agents, and other external organizations. Identify any special life cycle-related responsibilities assumed by the customer, or operational stakeholders, e.g.

- Performance of design or development functions
- Providing facilities, equipment, or software
- Supplying data
- Performing conversion and installation functions
- Supplying support services (computer operations, data entry, transportation, security, etc.)

Describe the following items for each stage in LCP 3.2.1, 3.2.2, and 3.2.3:

- **Organization**: Identify the major organizational components of the project and indicate their responsibilities during the various stages of the life cycle. Adapt Table 23 and elaborate OCD 3.6.2 as appropriate. Provide organization charts showing the major organizational components of the project during this stage and the key responsible personnel within each organizational component. Record any special agreements on organizational boundaries (e.g., the boundaries between organizations performing integration and system test; the boundaries between the organizations performing the functions of quality management, test and management)

- **Staffing**: Identify the types of personnel required by the project (analyst, programmer, operator, clerical, etc.) and the number of personnel of each type required to perform each major function during each life cycle phase. Identify critical skills required of the personnel (e.g., experience with Java, image compression, Sybase). Provide charts or tables showing the staffing requirements as a function of calendar time. Provide any special plans for hiring, relocation, security clearances, organizational transfers, use of temporary personnel, special compensation, etc.

- **Training**: Identify the organizations responsible for internal (project personnel) and external (customer, owner, user personnel) training. For both internal and external training, indicate the number of personnel to be trained, the length of training classes, the schedules for training preparation and performance, and the required facilities, equipment, software, instructors, training materials, etc.
Table 23: Stakeholder responsibilities during the software life cycle

<table>
<thead>
<tr>
<th></th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Users</strong></td>
<td>Support definition of requirements specification, operational concept and plan. Review prototype and exercise if available.</td>
<td>Review designs and prototypes during ARB. Help provide test data and test scenarios.</td>
<td>Review and test product (or its increment) in development environment. Provide test support.</td>
<td>Review and test product (or its increment) in operational environment. Provide usage feedback to Maintainer</td>
</tr>
<tr>
<td><strong>Customer</strong></td>
<td>Support definition and review of requirements specification, operational concept and plan – accept or reject options</td>
<td>Monitor progress at milestones. Review designs, prototypes, plans and feasibility during ARB. Help provide test data and test scenarios.</td>
<td>Monitor progress at milestones. Review and test product. Provide administrative support. Review system performance</td>
<td>Monitor progress. Provide administrative support in transitioning the product. Review system performance</td>
</tr>
<tr>
<td><strong>Developer/Maintainer</strong></td>
<td>Prepare requirements specification, operational concept, architectural sketches and plan. Build user interface prototype.</td>
<td>Refine architecture and design and present them during ARB. Refine or rebuild further prototype to investigate risks. Prepare test plan.</td>
<td>Refine design, implement, and integrate product. Perform and support reviews and test.</td>
<td>Provide development support in transitioning the product. Adapt product if development environment differs from operational one.</td>
</tr>
<tr>
<td><strong>Interfacer</strong></td>
<td>Support definition of requirements specification and interface specification.</td>
<td>Refine interface specification and review design. Build prototype to investigate risks.</td>
<td>Review interface design and implementation. Validate interface in development environment</td>
<td>Validate interface in operational environment</td>
</tr>
</tbody>
</table>

Common Pitfalls:
- Simply copying Table 23. You should adapt it to the particular project. Expect adaptations for COTS intensive systems, non-eServices systems, and eServices systems with significant front-end or back-end functionality.

3.1.1 Stakeholder Representatives

Identify by organizational position the personnel responsible for committing their stakeholder organization to any changes in project scope, budget, and schedule.

3.1.2 Engineering Stage Responsibilities

The normal project team responsibilities for the Engineering Stage will include task leaders responsible for the OCD, SSRD, SSAD, LCP, and Prototype, reporting to a project manager also responsible for the FRD. Identify any differences from this approach, plus any key client roles and responsibilities during the Engineering Stage. User and Customer responsibilities may include meetings/discussions regarding the new system, participating in prototyping, requirements gathering, architecture review boards, and execution and coordination of complementary initiative identified in the Results Chain (OCD 2.1)
3.1.3 Production Stage Responsibilities

User and Customer responsibilities may include development or modification of databases; training; parallel operation of the new and existing systems; impacts during testing of the new system; preparation for deployment in the organization; and other activities needed to aid or monitor development; and continuing execution and coordination of complementary initiatives identified in the Results Chain (OCD 2.1).

3.1.4 Support Stage Responsibilities

Operational Roles and Responsibilities can largely reference OCD 4.7.1. Ensure that maintenance roles and responsibilities are comparably defined.

3.2 Development Responsibilities

Describe a specific organization and set of team member roles and responsibilities. Calibrate the necessary team size using software effort and schedule estimation techniques. It should be detailed enough so that it could be used as a detailed construction plan, assigning tasks and responsibilities to team members. Team members should participate in defining and negotiating these. This will ensure performer commitment, help the team members work more efficiently, and help them finish their tasks quicker. It is also important to resolve any overlaps or underlaps between tasks assigned to different team members.

Below is an ordered list of neighboring roles and responsibilities, which can be clustered in various ways to best fit the project's situation:

- Project management, planning and control
- Client facilitation: operational procedures, transition planning and support, conversion and data preparation support, training, user manuals
- Item Qualification test and System Qualification test
- Quality Focal Point: Preparing and running independent testing (plans, cases); takes responsibility for the quality of the product and/or processes being used
- Configuration Management and document integration
- Detailed design, Programming, Unit Testing, Integration Testing, Maintenance Manuals
- Independent review and testing by the IV&V

577 Guidelines:

The Work Breakdown Structure is used to address team roles and responsibilities. Teams should tailor the WBS (LCP 5.1) and the Budget (LCP 5.2), before proceeding in this section.

3.2.1 Development Organization Charts

Provide a Development Organization Chart, and discuss any unusual features of it.

3.2.2 Staffing

Describe (as applicable):

1. The estimated staff-loading for the project (number of personnel over time). This plan will be tracked using effort metrics.
2. The breakdown of the staff-loading numbers by responsibility (for example, management, software engineering, software testing, software configuration management, software product assurance, software process assurance) and by software development item.

3. A breakdown of the skill levels expected or required of personnel performing each responsibility

The staffing plan should reference (as applicable) the work breakdown structure.

**Common Pitfalls:**

- Not identifying the staffing roles per WBS and instead only including total levels.

### 3.2.3 Training

Describe as applicable plans for getting the development team up to speed on critical skills.

### 4. Approach

This section identifies the activities to be performed in the life cycle and describes details of the approach that would be followed to create and manage the process, product, property and success models.

**Integration and Dependencies with other components:**

The Risk Management and Monitoring Procedures section is strongly coupled with the Project Risk Assessment section of the Feasibility Rationale Description (FRD 4.). In addition, since MBASE life cycle processes are risk-driven, the overall life cycle strategy in LCP 2.1 (incremental, design-to-schedule, etc.), will reflect the major risks. Product assessment and/or evaluation is closely coupled with the quality requirements in the SSRD. Facilities plans are coupled with the OCD. The section on Reviews focuses on the major milestones and deliverables, and is thus coupled with LCP 2. The milestone content also creates dependencies with the capabilities and requirements described in the SSRD. Once the capabilities have been refined in more detail, the milestone reviews will also address more detailed knowledge from the SSAD.

### 4.1 Monitoring and Control

Describe the techniques, procedures, and reports to be used in tracking project progress vs. plans and expenditures vs. budgets. For most projects, a straightforward closed-loop feedback control process with respect to your milestone plans (LCP 2.2), budgets (LCP 5.2), and risk management plans (LCP 4.1.4) will be sufficient. Highly complex projects may also include:

- Summary Task Planning Sheets
- Earned Value Status Reports
- Project Expenditure Summaries
- Cumulative Milestone Progress Reports
- PERT/ COST Systems
- Budget-Schedule-Milestone Charts
- Personnel Loading Charts
- Detailed Expenditure vs. Budget Reports
Guidelines for MBASE

- Control Limit Monitoring
- Statistical Process Control

Provide

- Links to Periodic Progress Reports
- Links to Effort Reports
- Links to Other Monitoring and Control Artifacts

577 Guidelines:

Most of the items above would be an overkill for CS577. It will generally be sufficient to identify how milestones will be tracked (e.g., via text schedules, Microsoft Project, etc.), and who is responsible for monitoring and controlling what (e.g., project manager for major milestone completion, Quality Focal Point for overall product quality, including peer reviews and product content).

Describe the software metrics used for tracking and controlling the project development, and the process used to collect and analyze the metrics. Each team must report weekly progress, effort and trouble report metrics as well as risk items, using Weekly Effort Forms submitted by the team members. Describe all the different sets of progress metrics that will be tracked. Examples include major development milestones, lines of code, etc. Progress metrics can also be broken down by function or subteams. Optional metrics include requirements volatility.

4.1.1 Closed Loop Feedback Control

Closed loop feedback control involves monitoring your progress and resource expenditures in your plans and budgets, and determining when these begin to deviate significantly. When this happens, it involves applying corrective action to bring the projects status, plans, and objectives back into a feasible state.

4.1.2 Reviews

This section identifies the major project reviews and their objectives. It provides the plans to prepare for, conduct, and follow up on the review meeting in order to achieve the objectives of each review.

The primary objective of each major review is to determine whether or not the project is ready to proceed to the next life cycle phase. If so, the phase products reviewed are baselined and put under configuration management, to provide a stable foundation for the following phase. Note that the LCO package is baselined, even though the high priority is to evolve it into an LCA version.

Preparing the Reviews section often requires a good deal of tailoring. Each project review is actually a small project in itself, and should thus have its own small project plan, indicating its objectives, milestones, responsibilities, approach, resources, and assumptions.

In practice, particularly on small projects, there may be a good deal of overlap between these sections (e.g., between "milestones" and "approach"), and between the plans for the various reviews. In such cases, it is best to compress LCP 4.1.2 into a single generic plan, accompanied by a table indicating the unique characteristics of each review.

Review plans should emphasize the following activities.

- **Before:**
  - getting user, customer, and other success-critical stakeholders organizations to participate;
  - securing commitments from capable reviewers;
• preparing review assignments;
• distributing review material well in advance;
• getting itemized written comments from reviewers;
• providing the comments to the developer, and getting the developer to prepare his or her response;
• setting up the review meeting agenda;

• After:
• publishing review meeting minutes, documenting agreements reached at the meeting;
• assigning, tracking, and closing out action items from the review meeting;

The review materials should include not only the phase products, but also evidence of the developer's having verified and validated them (e.g., the Feasibility Rationale Description in the LCO and LCA packages).

Subsections of LCP 4.1.2 are indicated below for the reviews corresponding to the nominal phase organization in Table 24. These subsections should be modified to reflect any significant departures of the project's development strategy from this nominal approach.

Table 24: Key Products & Technical Management Reviews (Excluding In-Process Peer Reviews)

<table>
<thead>
<tr>
<th>Phases:</th>
<th>Inception (LCO)</th>
<th>Elaboration (LCA)</th>
<th>Construction (IOC)</th>
<th>Transition</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management Reviews:</td>
<td>ARB-I</td>
<td>ARB-II</td>
<td>TRR</td>
<td>RRR</td>
<td></td>
</tr>
<tr>
<td>Package Elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational Concept Description (OCD)</td>
<td>▲</td>
<td></td>
<td>▲</td>
<td>▲</td>
<td></td>
</tr>
<tr>
<td>System and Software Requirements Definition (SSRD)</td>
<td>▲</td>
<td>▲</td>
<td></td>
<td>▲</td>
<td></td>
</tr>
<tr>
<td>System and Software Architecture Description (SSAD)</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Life Cycle Plan (LCP)</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td></td>
</tr>
<tr>
<td>Feasibility Rationale (FRD)</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype</td>
<td>▲</td>
<td>▲</td>
<td></td>
<td>▲</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>▲</td>
</tr>
<tr>
<td>Deployment Set</td>
<td></td>
<td></td>
<td>▲</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transition Plan</td>
<td></td>
<td>▲</td>
<td>▲</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: this table is adapted from [Royce, 1998] Figure 6-10. The dark bullets are controlled (strong) baselines whereas the light bullets are informal (weaker, less defined) baselines.
### Guidelines for MBASE Life Cycle Plan (LCP)

#### Table 25: Key Products & Technical Management Reviews (Excluding In-Process Peer Reviews) For CS577ab

<table>
<thead>
<tr>
<th>Phases: Anchor points</th>
<th>Inception (LCO)</th>
<th>Elaboration (LCA)</th>
<th>Construction (IOC)</th>
<th>Transition (IOC)</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management Reviews:</td>
<td>ARB-I</td>
<td>ARB-II</td>
<td>ARB-III</td>
<td>TRR</td>
<td>RRR</td>
</tr>
<tr>
<td>Package Elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational Concept Description (OCD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System and Software Requirements Definition (SSRD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System and Software Architecture Description (SSAD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Life Cycle Plan (LCP)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feasibility Rationale (FRD)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transition Plan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The dark circles are controlled strong baselines whereas the gray circles are the informal baselines and the light circles are weaker less defined baselines.

**577 Guidelines:**

General principles for ARB reviews are available in [AT&T, 1993]. ARB reviews need to put review materials on the Web a week in advance, and to arrange a satisfactory review time for their clients. Reviews involve short highlight presentations by each team member, including a prototype demo. A review scribe should summarize the review results, with associated actions, agents, and due dates.

- You may reference plans and reports which will be done in the Production Stage
- Review Plan
- Peer Review Plan
- Peer Review Reports
- Review plan should indicate how client wants to accomplish the reviews (one vs. more meetings; demos; training sessions, etc.) and identify candidate dates for each.
- Architecture Review Board I (ARB-I)
  - The Review Criteria for the LCO Architecture Review are the completion criteria for LCO Feasibility Rationale.
• Architecture Review Board II and III (ARB-II and ARB III)
  • The Review Criteria for the LCA Architecture Review are the completion criteria for LCA Feasibility Rationale.

• In-Process Peer Reviews
  • Detailed Design Inspection or Review
    • Code Inspection or Review
    • User Manual Inspection or Review
  
  Refer to the inspection or peer review guidelines on the class web site, and their associated inspection or peer review reports. See also [CMU-SEI, 1995], Chapter 8.7.

• Transition Readiness Review (TRR)
  • The Transition Readiness Review (TRR) should verify that the following transition pre-conditions are satisfied:
    • Ready-to-install software, verified for compliance with the requirements in the SSRD;
    • Ready-to-use User's Manual, Maintenance Manual, training material, installation and operational procedures;
    • Ready-to-use client facilities, equipment, software infrastructure, and applications data;
    • Committed client personnel for transition and training

  The Transition Plan may identify transition preconditions whose satisfaction is deferred for the Release Readiness Review (e.g., final tailoring of user interfaces and operational procedures). The completion criteria for the Release Readiness Review (RRR) will include the completion criteria for the deliverables in Section 2.3, plus any situation-specific criteria (e.g., the degree of cutover from the existing operation to be accomplished by the Product Readiness Review).

• Release Readiness Review (RRR)
  • The Release Readiness Review (RRR) should verify the successful completion of the Transition Plan and the Readiness of the system and the clients to transition into client operations. The Release Readiness Review (RRR) should review all operations-critical items, such as:
    • system preparation;
    • training;
    • usage;
    • evolution support with clients.
  
  It should reference section 2.3 for “acceptability” criteria for deliverables
• Review plan should indicate how client wants to accomplish the final review to assure satisfactory system transition and identify candidate dates for each

Common Pitfalls:

• Do not simply duplicate Table 25 as your key products and reviews model. The table is a guideline only and its elements should be tailored to your project and explicitly accounted for and planed.

4.1.3 Status Reporting

Between the major phase reviews, the project manager should prepare and deliver periodic status reports summarizing progress with respect to plans and budgets, and associated corrective actions. The frequency and mode (document, meeting, video conference, etc.) of these reports should be determined by risk considerations. To facilitate closed loop control, the status reporting should track the number of avoidable issues, problem reports and defects.

577 Guidelines:

Weekly written reports are used for CS577. The report MUST include the number of avoidable issues, problem reports and defects which have been carried into the reporting period, identified within the reporting period, and finally the number closed within that period [leaving the number to be carried forward to the next report].

4.1.4 Risk Monitoring and Control

Document procedures for monitoring the risk factors and for reducing the potential occurrence of each risk. Identify contingency procedures for each area of risk. Show how the project will address, monitor, and resolve these sources of risk, by providing plans for mitigating the identified risks

• Monitoring of risk management plan milestones
• Corrective Action: develop or invoke Contingency Plans
• Top-10 Risk Item Tracking (See Table 26)
  • Identify top 10 Risk Items
  • Highlight these in regular project reviews (focuses review on manager-priority items)
  • Focus on new entries and slow-progress items
• Risk Reassessment

<table>
<thead>
<tr>
<th>Risk Items</th>
<th>Weekly Ranking</th>
<th>Risk Resolution Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current</td>
<td>Previous</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
For each critical risk item, develop a detailed risk management plan, and provide a link to it below. A Risk Management Plan answers the following questions:

1. **Objectives (Why? Whereas?):** Risk item importance, relation to project objectives, underlying assumptions
2. **Deliverables and Milestones (What? When?):** Risk resolution deliverables, milestones, activity nets
3. **Responsibilities and Organization (Who? Where?):**
4. **Approach (How?):** Prototypes, surveys, models, …
5. **Resources (How Much?):** Budget, schedule, key personnel, …

**Common Pitfalls:**
- Describing the project risks in detail here, they belong to the FRD 4.

**Provide**
- Links to Individual Risk Management Plans

**577 Guidelines:**
A risk management technique commonly used is a weekly top-10 Risk Items Lists

**4.1.5 Project Communication**
Provide a plan for how team members will communicate with each other and the client(s) covering:
- Meeting schedules
- Use of email, web, etc.
- Conference calls, etc.
- Use of word processing systems for document integration

**4.2 Methods, Tools and Facilities**
Identify, as appropriate, the functions, milestones, responsibilities, physical configurations and operational procedures involved in preparing and operating project facilities, and in handling related concerns, including:
- Support services
- Support software
- Customer furnished facilities, equipment, and services
- Security
- Subcontractor operations
- Use of commercial software

Facilities may include:
577 Guidelines:

Identify equipment, software, services, documentation, data, and facilities provided by USC. A schedule detailing when these items will be needed shall also be included. Also, include other required resources, including a plan for obtaining the resources, dates needed, and availability of each resource item.

4.3 Configuration Management

577 Guidelines:

Upto LCA time, it will be sufficient to identify:

- Which items will be baselined when
- How changes to the baseline will be coordinated with the client (e.g., meeting for major changes, email for moderate changes, none for trivial changes)
- How outstanding problem reports will be tracked
- Who will be the custodian of the master baselined versions, and how he/she will preserve the integrity and recoverability of the master versions
- Who will be the Configuration Manager and how will he/she ensure that a version control is maintained.

At Rebaselined LCA time:

The majority of this information should be in the Quality Management Plan.

At IOC time:

The rest of this section will be replaced in the next major revision of the guidelines.

4.3.1 Product Element Identification

The product elements to be identified should include the project Deliverables enumerated in LCP 2.3, as appropriate, and any other elements that stakeholders may consider important (e.g. operational procedures, project critiques). Each product element should have an owner and a unique identifier. Each individual sub element of the OCD, SSRD, SSAD, and delivered software package should have a unique identifier, enabling traceability relations to be established among these sub elements.

Those product elements to be placed under baseline configuration management (CM) should be identified, along with the project milestones at which they enter the CM process. Table 25 in Section 4.1.2 provides a recommended set of elements to be baselined and their associated milestones.

Provide
Guidelines for MBASE Life Cycle Plan (LCP)

- Links to Product Element Identification Directories

577 Guidelines:

It is sufficient to mention which product elements will be considered as strong baselines along which project milestone.

Note: Strong baselines are those changes that are suggested by the client that if made could affect the design and the system significantly. The details of these changes can be viewed in the artifacts version history or the configuration history and the minutes of the meetings between the team and the client. These can also be viewed from the review forms that clients use to log multiple concerns.

4.3.2 Configuration Change Management

Provide a flow chart indicating the sequence of tasks and decisions involved in submitting, analyzing, approving, and implementing proposed changes to software baseline items. Provide the associated change control forms and procedures, and a chart indicating what level of management is responsible for approving the various classes of proposed changes.

Provide

- Links to Change Management Procedures

577 Guidelines:

- At Rebaselined LCA time: This majority of the information in this section should be in the Quality management Plan in Section 2.5.4 Configuration Control.

4.3.3 Project Library Management

Describe the operation of the project library, including:

(a) organizational responsibilities
(b) library contents
(c) services provided
(d) operational procedures for general usage, storage and release of master copies, security, backup and recovery
(e) library facilities and support services
(f) staffing and resource requirements

Provide

- Links to Project Library

577 Guidelines:

The weak/informal baselined project elements along different project milestones, if any, need to be identified here. Also any of the other project elements that are not configured but need to be stored are to be mentioned in this section.

Note: The weaker less defined baselines are those that can be changed by the team members independently. The informal baselines, are the changes that are suggested by the ARB, peer-reviews etc that are discussed by the team.
or informally with the client. The details of these changes can be viewed in the artifacts version history and change summary or the configuration history.

4.3.4 Configuration Status Management

Identify the responsibilities, schedules, and procedures involved in performing audits of integrity of the product elements.

Provide

- Links to Status Accounting System

577 Guidelines:

It is sufficient to mention how changes at different baselines (strong, informal, weak) be handled by the team and where the change information can be retrieved.

4.4 Quality Management

The main tasks in Quality Management are quality assessment (and repair), quality tracking (and repair) and process improvement(s) to reduce the occurrence of "defects", increase the effectiveness of "defect" identification, or apply lessons-learned to the current project or future projects. Another aspect of quality management is whether it is being applied to product(s) such as documents and code, which is often called "product assurance", or it is being applied to processes, often called "process assurance". Finally, not all components of a product might be expected to have the same level of product quality. Different artifacts or code items might be treated differently on an overall project risk driven bases.

577 Guidelines:

At LCO time, it is sufficient to identify:

- Which quality assessment activities have been carried out, and where the resulting reports can be found. This can be accomplished with a simple table that identifies what quality assessment techniques were applied when to what documents, and where the resulting "quality" reports can be found (by reference preferred).

<table>
<thead>
<tr>
<th>Technique</th>
<th>Applied When</th>
<th>What Document</th>
<th>Quality Reports Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fagan's Inspection</td>
<td>Week of 10/8/01</td>
<td>OCD Early Sections see .../team21a/LCO/EC_OCD1_LCO_F01a_T21.doc</td>
<td><a href="http://...team21a/LCO/QualityReports/OCD1_LCO_IR_F01a_T21.xls">http://...team21a/LCO/QualityReports/OCD1_LCO_IR_F01a_T21.xls</a></td>
</tr>
<tr>
<td>Fagan's Inspection</td>
<td>10/25/01</td>
<td>SSAD v.0.1</td>
<td><a href="http://...team21a/LCO/QualityReports/SSAD_LCO_IR_F01a_T21.xls">http://...team21a/LCO/QualityReports/SSAD_LCO_IR_F01a_T21.xls</a></td>
</tr>
</tbody>
</table>

- What quality assessment activities are planned to be carried out during Elaboration (at least the first cycle of the next phase). A simple tabular list that identifies what quality assessment techniques are to be applied when, and to which documents, will suffice.

- What is planned for the Construction phase and where it will be documented. This can be very general.

- How outstanding problem reports and open issues will be tracked.

At LCA time, it is sufficient to identify:
• Which quality assessment activities have been carried out, and where the resulting reports can be found. This can be accomplished with a simple table that identifies what quality assessment techniques were applied when to what documents, and where the resulting "quality" reports can be found (by reference preferred). The same format that was used for Inception is acceptable.

• What quality assessment activities are planned for Elaboration (at least the next cycle of this next phase) and Construction? A simple tabular list that identifies what quality assessment techniques are to be applied, when, and to which documents, will suffice.

• What is planned for the Construction phase and where it will be documented? This can be very general.

• How outstanding problem reports and open issues will be tracked.

At Rebaselined LCA time and IOC, it should contain:

• The history of which quality assessment activities have been carried out and when, and where the resulting reports can be found. The same format that was used for Inception is acceptable.

• A reference to the Quality Management Plan for the expansion of the majority of the information only identified at LCO or LCA.

4.4.1 Process Assurance

The objective of software quality management is to ensure the delivery of a high-quality software product by determining the project’s prioritized quality objectives and verifying that the project's agreed-upon plans, policies, standards, and procedures for achieving those objectives are all adhered to.

This section should elaborate on Level of Services, and roles & responsibilities of team members in achieving them. As emphasized in [Royce, 1998], quality is everyone's responsibility, but it is still generally useful to include traditional quality assurance functions such as:

a) Auditing the project's compliance with its plans, policies, and procedures;

b) Monitoring the performance of reviews and tests;

c) Monitoring corrective actions taken to eliminate reported quality related deficiencies

577 Guidelines:

Due to the significant schedule constraints, no explicit process assurance related activities are done by the teams. The instructional staff does provide some process assurance in terms of scheduling deliverables, tasks, assignments, and carryover of lessons learned through changes in the course and/or the MBASE Guidelines.

4.4.2 Product Assurance

One of the prime objectives of software quality management is to ensure the delivery software products of an acceptable level of quality that also meets the client requirements and expectations.

This section should elaborate on Level of Services, and roles & responsibilities of team members in achieving them. As emphasized in [Royce, 1998], quality is everyone's responsibility, but it is still generally useful to include traditional quality assurance functions such as:

a) Development of documentation and code standards

d) Verification of the project's compliance with its documentation and code standards;

b) Requirement specification, design and architecture.
Guidelines for MBASE Life Cycle Plan (LCP)

c) Coding

d) Development of Test Plans, Quality Plans, Transition Plan etc

e) Testing

f) Verification & Validation

And all such related activities that are involved in product development.

Common Pitfalls:

Identify just the key subset of standards the team should follow and compliance-manage (e.g., header block information for code modules; document formats; avoidance of error-prone code constructs, exotic OS features limiting portability, etc.).

Do not promise more than you want to deliver. Follow the rule: If it’s risky to include an item, don’t put it in. If it’s risky not to put an item, do put it in. Identify all the product deliverables that will be reviewed and to what detail.

577 Guidelines:

During the engineering phase, it is sufficient to reference the LCP section 2.2 Phase Milestones and Schedules for planned reviews and test activities. During the production phase, all of the details about reviews and tests should be in the Quality Management Plan.

The Verification & Validation aspects of Product Assurance are covered separately in LCP section 4.4.2.1 Verification and Validation.

4.4.2.1 Verification and Validation

Verification is “the process of evaluating a system or component during or at the end of the development process to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase” [IEEE Standards Board 1990]. Validation is “the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies [the needs of the customer]” [IEEE Standards Board 1990].

Verification and validation analyze the specification of the requirements, the design and the implementation using techniques such as modeling, prototyping, simulation, reviews, and testing. Testing includes unit testing, integration testing, item qualification testing and system qualification testing. In MBASE, the verification and validation results are documented in the Section 2.4 of the Quality Management Plan.

Often the customer hires another experienced company or individual to independently verify and validate (“IV&V”) the work of the developing organization. There is a tendency among developers to downplay the effect of problems and concerns, and to exaggerate the effect of features. The IV&V company or individual can check such tendencies and provide the customer with greater assurance that the developers are producing something that will meet the customer’s needs.

577 Guidelines:

Every member in the team is responsible for verification and validation in the form of peer reviews and testing.

The team also has a Quality Focal Point who is responsible for internal verification and validation of the project and the overall quality of the product.

Each team is assigned an IV&V person. His/her tasks include performing independent testing and reviews. The IV&V being an external reviewer needs to review and test all deliverables for completeness and correctness and ensure the verification of the project’s compliance with its documentation and code standards and feasibility of the solution.
5. Resources

This section tells *how much* resources the project will require to perform the functions indicated in the previous sections. It identifies the resources by type (personnel, calendar time, capital dollars, operations dollars, etc.), by expenditure category (labor, computer, travel, publications, etc.), by life cycle phase, and by project activity.

**Integration and Dependencies with other components:**

The tasks must fit into the required capabilities as described in the SSRD and SSAD, which are summarized in LCP 5.1. Based on that summary, the budget can be estimated and refined. The resource requirements are the levels of investment needed for the business case in the Feasibility Rationale Description.

### 5.1 Work Breakdown Structure

The WBS provides a hierarchical ordering of project tasks and activities which serves as a basis for project budgeting, cost collection, and control.

**Tailor** the following WBS chart indicating the project's WBS elements, their associated budgets, and the person responsible for the tasks and budgets. Form the WBS by an appropriate tailoring of the project-specific WBS product.

<table>
<thead>
<tr>
<th>A</th>
<th>Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>Inception phase management</td>
</tr>
<tr>
<td>AAA</td>
<td>Top-level Life Cycle Plan (LCO version of LCP)</td>
</tr>
<tr>
<td>AAB</td>
<td>Inception phase project control and status assessments</td>
</tr>
<tr>
<td>AAC</td>
<td>Inception phase stakeholder coordination</td>
</tr>
<tr>
<td>AAD</td>
<td>Elaboration phase commitment package and review (LCO package preparation and ARB review)</td>
</tr>
<tr>
<td>AB</td>
<td>Elaboration phase management</td>
</tr>
<tr>
<td>ABA</td>
<td>Updated LCP with detailed construction plan (LCA version of LCP)</td>
</tr>
<tr>
<td>ABB</td>
<td>Elaboration phase project control and status assessments</td>
</tr>
<tr>
<td>ABC</td>
<td>Elaboration phase stakeholder coordination</td>
</tr>
<tr>
<td>ABD</td>
<td>Construction phase commitment package and review (LCA package preparation and ARB review)</td>
</tr>
<tr>
<td>AC</td>
<td>Construction phase management</td>
</tr>
<tr>
<td>ACA</td>
<td>Updated LCP with detailed transition and evolution plans</td>
</tr>
<tr>
<td>ACB</td>
<td>Construction phase project control and status assessments</td>
</tr>
<tr>
<td>ACC</td>
<td>Construction phase stakeholder coordination</td>
</tr>
<tr>
<td>ACD</td>
<td>Transition phase commitment package and review (IOC package preparation and PRB review)</td>
</tr>
<tr>
<td>AD</td>
<td>Transition phase management</td>
</tr>
<tr>
<td>ADA</td>
<td>Updated LCP with detailed next-generation planning</td>
</tr>
<tr>
<td>ADB</td>
<td>Transition phase project control and status assessments</td>
</tr>
<tr>
<td>ADC</td>
<td>Transition phase stakeholder coordination</td>
</tr>
<tr>
<td>ADD</td>
<td>Evolution stage commitment package and review (PR package preparation and PRB review)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>Environment and Configuration Management (CM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
<td>Inception phase environment/CM scoping and initialization</td>
</tr>
<tr>
<td>BB</td>
<td>Elaboration phase environment/CM</td>
</tr>
<tr>
<td>BBA</td>
<td>Development environment installation and administration</td>
</tr>
<tr>
<td>BBB</td>
<td>Elaboration phase CM</td>
</tr>
<tr>
<td>BBC</td>
<td>Development environment integration and custom toolsmithing</td>
</tr>
<tr>
<td>BC</td>
<td>Construction phase environment/CM evolution</td>
</tr>
<tr>
<td>BCA</td>
<td>Construction phase environment evolution</td>
</tr>
<tr>
<td>BCB</td>
<td>Transition phase CM</td>
</tr>
</tbody>
</table>
Guidelines for MBASE Life Cycle Plan (LCP)

BD Transition phase environment/CM evolution
BDA Construction phase environment evolution
BDB Transition phase CM
BDC Evolution stage environment packaging and transition

C Requirements
CA Inception phase requirements development
CAA Operational Concept Description and business modeling (LCO version of OCD)
CAB Top-level System and Software Requirements Definition (LCO version of SSRD)
CAC Initial stakeholder requirements negotiation

CB Elaboration phase requirements baselining
CBA OCD elaboration and baselining (LCA version of OCD)
CBB SSRD elaboration and baselining (LCA version of SSRD)

CC Construction phase requirements evolution
CD Transition phase requirements evolution

D Design
DA Inception phase architecting
DAA Top-level system and software architecture description (LCD version of SSAD)
DAB Evaluation of candidate COTS components

DB Elaboration phase architecture baselining
DBA SSAD elaboration and baselining
DBB COTS integration assurance and baselining

DC Construction phase design
DCA SSAD evolution
DCB COTS integration evolution
DCC Component design

DD Transition phase design evolution

E Implementation
EA Inception phase prototyping
EB Elaboration phase component implementation
EBA Critical component implementation

EC Construction phase component implementation
ECA Alpha release component coding and stand-alone testing
ECB Beta release (IOC) component coding and stand-alone testing
ECC Component evolution

ED Transition phase component evolution

F Assessment
FA Inception phase assessment
FAA Initial assessment plan (LCO version; part of LCP)
FAB Initial Feasibility Rationale Description (LCO version of FRD)
FAC Business case analysis (part of FRD)
FAD Peer Review (information in LCP)

FB Elaboration phase assessment
FBA Elaboration of assessment plan (LCA version; part of LCP)
FBB Elaboration feasibility rationale (LCA version of FRD)
FBC Elaboration of business case analysis (part of FRD)
FBD Elaboration of Peer Review (information in LCP)
FBE Development of test plans

FC Construction phase assessment
FCA Elaboration of assessment plan approaches (part of QMP)
FCB Detailed test plans and procedures (part of QMP, TP, TD&R)
FCC Evolution of feasibility rationale
FCD Peer Reviews (part of QMP, PRP, RRR)
FCE Alpha release assessment
FCF Beta release (IOC) assessment

FD Transition phase assessment

G Deployment
577 Guidelines:

The WBS in [Royce, 1998] Section 10.1 is good for big projects but an overkill for 577 projects. It is generally sufficient to construct a WBS as an organization chart, identifying the number of full-student-time-equivalent people doing which functions (e.g., 2.0 for two team members devoted to programming, down to identifying each programmer's software components), plus any WBS elements for equipment costs, data preparation costs, etc.

Common Pitfalls:

- Do not duplicate the WBS outline as your deliverable. Choose only elements within it that apply to your project and detail it as described in the guidelines.

5.2 Budgets

Provide breakdowns of the software life cycle project budget and staffing level requirements. These should include, as appropriate, breakdowns by:

- WBS element;
- Phase and by calendar month;
- Labor grade (analyst, programmer, operator, clerical, etc.);
- Budget category (capital dollars, operations dollars, etc.);
- Expenditure category (labor, computer, travel, publications, miscellaneous).

A checklist of potential miscellaneous expenditures is given in Table 28.

Include Effort and Schedule estimates using at least two different, credible, and repeatable cost estimation techniques.

- Object points (You should have more or less a fair idea on the number of screens, reports, and 3GL components in your application, especially after doing your prototype, SSRD, and SSAD)
- SLOC (source lines of code), in whatever programming or scripting languages are used
- Function Points (backfiring tables can convert function-points to SLOC)
- Performer-based task-level estimates

It is critical to document any assumptions used to come up with the estimate: e.g., when using a software cost model, provide a rationale for the Effort Multiplier and Scale factor ratings and sizing parameters. The Effort (Person-months) and Schedule should be used to validate the staffing requirements and confirm that the project is feasible within the allocated budgets and schedule (FRD 3.3).
Table 28: Miscellaneous Software Project Expenditure Sources

<table>
<thead>
<tr>
<th>Cost Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clerical Costs</td>
<td>Overtime, benefits, hiring, termination, relocation, education; personnel costs for product acquisition: contracts, legal, receiving, peer review, etc.</td>
</tr>
<tr>
<td>Related Personnel Costs</td>
<td>Overtime, benefits, hiring, termination, relocation, education; personnel costs for product acquisition: contracts, legal, receiving, peer review, etc.</td>
</tr>
<tr>
<td>Related Computer Costs</td>
<td>Installation, maintenance, insurance, special equipment: terminals, control units, data entry devices, etc.</td>
</tr>
<tr>
<td>Office Equipment Costs</td>
<td>Computers, telephones, copiers, file cabinets, desks, chairs, software, etc.</td>
</tr>
<tr>
<td>Software Product Costs</td>
<td>Purchase, rental, licensing, maintenance of software components, utilities, tools, etc.</td>
</tr>
<tr>
<td>Supplies Costs</td>
<td>Disks, forms, paper, print cartridges, office supplies, etc.</td>
</tr>
<tr>
<td>Telecommunication Costs</td>
<td>Line charges, connection charges, special equipment: modems, video conferencing, etc.</td>
</tr>
<tr>
<td>Facility Costs</td>
<td>Office Rental, electricity, air conditioning, heating, water, taxes, depreciation, cleaning, repairs, insurance, security, fire protection, etc.</td>
</tr>
<tr>
<td>Other costs</td>
<td>Travel, postage, printing, consulting fees, books, periodicals, conventions, equipment relocation, etc.</td>
</tr>
</tbody>
</table>

Common Pitfalls:

Inconsistent with SSRD budget requirements (SSRD 2.1) and FRD development costs (FRD B.2.1.1)

577 Guidelines:

The budget breakdowns in the guidelines are more for big projects than for class projects. Whether you are using COCOMO II or some other method, provide a manual Object-Points analysis and a SLOC-based estimate, and provide breakdowns by item of the overall equipment, data preparation and other costs identified in the WBS in Section 5.1.

6. Appendices

The Appendix may be used to provide additional information published separately. As applicable, each appendix shall be referenced in the main body of the document where the data would normally have been provided.

Properly formatted COCOMO Results including the CLEF, EAF for each module, schedule and scale factors and phase distributions.

- Detailed Gantt Charts for milestones and schedules
Feasibility Rationale Description (FRD)

A. Description

1. Purpose

The purpose of the Feasibility Rationale Description (FRD) is to ensure that the system developers have not just created a number of system definition elements, but have also demonstrated the feasibility and consistency of these elements. The FRD contents should:

- Ensure feasibility and consistency of the other system definition components (OCD, SSRD, SSAD, LCP, Prototype)
- Demonstrate a viable business case for the system
- Identify shortfalls in ensuring feasibility, consistency, and business case as project risk items for LCP
- Demonstrate that a system built using the specified architecture (described in the SSAD) and life cycle process (described in the LCP) will:
  - satisfy the requirements described in the SSRD
  - support the operational concept described in the OCD
  - satisfy the success-critical stakeholders in the OCD and LCP
  - remain faithful to the key features determined by the prototype described in the OCD and SSRD
  - stay within the budgets and schedules in the LCP
- Rationalize life cycle decisions in a way the prime audience (the customer and users) and other stakeholders can understand
- Enable the success-critical stakeholders to participate in the decision process and to record their satisfaction with the product and their commitment to fulfill their roles as defined in the LCP.

2. Completion Criteria

Below are the completion criteria for the Feasibility Rationale Description for the three major milestones:

- Life Cycle Objectives (end of Inception Phase)
- Life Cycle Architecture (end of Elaboration Phase)
- Initial Operational Capability (end of Construction Phase)

2.1 Life Cycle Objectives (LCO)

- Assurance of consistency among the system definition elements above for at least one feasible architecture (this should be done via appropriate combinations of analysis, measurement, prototyping, simulation, modeling, reference checking, etc.)
• Assurance of a viable business case analysis for the system as defined
• Assurance that all success-critical stakeholders are committed to support the project throughout its Elaboration phase

2.2 Life Cycle Architecture (LCA)
• Assurance of consistency among the system definition elements above for the architecture specified in the SSAD
• Assurance of a viable business case analysis for the system as defined
• Assurance that all success-critical stakeholders are committed to support the project throughout its life cycle
• Assurance that all major risks are either resolved or covered by a risk management plan

2.3 Initial Operational Capability (IOC)
• Feasibility rationale for future increments beyond IOC
• Validation of business case and Results Chain (OCD 2.1) assumptions

3. Intended Audience
• The primary audiences are the LCO and LCA Architecture Review Board members, and all TRR and RRR reviewers, including:
  • Key system stakeholders;
  • Experienced peers;
  • Technical Specialists in critical areas.
• The parts dealing with client satisfaction must be understandable by the client representatives on the ARB.
• The technical parts must be sufficiently detailed and well organized to enable the peers and technical experts to efficiently assess the adequacy of the technical rationale.
• The FRD is of considerable future value to developers and other stakeholders in providing a rationale for important decisions made by the project.

4. Participating Agents
• All stakeholders are responsible for consistency and feasibility of the system definition via their participation in Win-Win negotiations and ARB reviews.
• Agreements can be contingent on demonstration of feasibility or risk resolution.

5. Performing Agents
• The project manager is responsible for the FRD content.
• The OCD author is usually the best-prepared person to develop the business case
6. High Level Dependencies

The thoroughness of the Feasibility Rationale Description is dependent on the thoroughness of all the other system definition elements. Any issues incompletely covered in the Feasibility Rationale Description are sources of risk, whose management should be covered in the Life Cycle Plan’s (LCP) Risk Management and Monitoring Procedures section (LCP 4.1).

7. Overall Tool Support

Well-calibrated estimation models for cost, schedule, performance, or reliability are good sources of feasibility rationale. Others are prototypes, simulations, benchmarks, architecture analysis tools, and traceability tools. The rationale capture capability in the Win–Win tool is also useful.

8. Degree of Detail and Tailoring

The degree of details of the FRD should be risk-driven (as with any MBASE model). If it’s risky to put an item in (e.g., unreliable cost or effort estimates, speculative value propositions), don’t put it in. If it’s risky not to put an item in (e.g., assessment or risks with potentially critical consequences), do put it in. Sections of the FRD may be tailored down, or consolidated for small or non-critical, well defined systems. Satisfaction of Capability and Interface requirements already documented in the SSAD should be referenced rather than repeated in FRD 2.2.3 and 2.2.4.

B. Document Sections

1. Introduction

1.1 Purpose of the Feasibility Rationale Description Document

- Summarize the purpose and contents of this document with respect to the particular project and people involved.

- Avoid generic introductions as much as possible: for instance, you can show how your particular Feasibility Rationale Description meets the completion criteria for the given phase.

Common Pitfalls:

- Simply repeating the purpose of the document from the guidelines

1.2 References

Provide complete citations to all documents, meetings and external tools referenced or used in the preparation of this document.

577 Guidelines:

A "complete citation" for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]
1.3 Change Summary

577 Guidelines:
For versions of the FRD since the last ARB, include a summary of changes made in the document to ease the review process.

2. Product Rationale

This section furnishes the rationale for the product being able to satisfy the system specifications and stakeholders (e.g. customer, user). It should also provide the rationale as to why the proposed system is better than the current system.

Integration and Dependencies with other components:
This section is highly dependent on all other documents. The cost estimates in FRD 2.1 are strongly dependent on development cost (from LCP) and operational cost (from OCD). FRD 2.2 maps requirements to design, which create a high dependency between the System and Software Requirements Description (SSRD), the System and Software Architecture Description (SSAD), and often the prototype. It creates a dependency between the OCD, the SSAD, and often the prototype. The stakeholder concurrence in FRD 2.3 summarizes the findings so that a green light can be given to proceed with the development.

2.1 Business Case Analysis

This section analyses the product’s return on investment: \( \text{ROI} = \frac{\text{Benefits} - \text{Costs}}{\text{Costs}} \).

- Costs Include development, transition, operations, and maintenance costs.
- Where possible benefits are expressed in financial terms compared to costs, such as increased sales and profits, or reduced operating costs.
- However, non-financial factors may be also decisive. For instance, “added value” can include the improved quality of the service provided by the product.
  - For a commercial system, the business case analysis will generally demonstrate an acceptable financial return on investment.
  - For a public service system, (health, education, defense, emergency services) the rationale would be expressed in terms of improvements in public service effectiveness as determined by analysis expressed by the users; or in terms of cost savings to achieve the desired level of effectiveness.

2.1.1 Development Cost Analysis

- Using estimates computed in the Budgets section (LCP 5.2), provide a summary of the full development cost, including hardware, software, people, and facilities costs.

Common Pitfalls:
- Repeating the analysis from LCP 5.2. Provide only a summary, and reference the detailed analysis

2.1.2 Transition Cost Estimate

- Provide a rough estimate of costs to be incurred during the transition of the product into production.
These costs may include:

- Training Time (trainer preparation and delivery time; trainee time)
- Data preparation
- COTS licenses
- Operational readiness testing
- Site preparation
  - Facilities preparation
  - Equipment purchase

### 2.1.3 Operational Cost Estimate

- Provide a summary of the operational costs, including license costs for COTS software, system administration costs, and database administration costs

### 2.1.4 Maintenance Cost Estimate

- Provide a summary of hardware and software maintenance costs if applicable. Life cycle cost models such as COCOMO II can be helpful.

**Common Pitfalls:**

- Repeating the analysis from LCP 5.2. Provide only a summary, and reference the detailed analysis

### 2.1.5 Estimate of Value Added and Return on Investment

- Provide a summary of cost with and without the product, and how much value it adds
- The value added may also describe non-monetary improvements (e.g. quality, response time, etc.), which can be critical in customer support and satisfaction.
- Non-monetary, qualitative value added by the system may often be estimated by considering how well the proposed system helps achieve the Organization Goals (OCD 3.2). This is why measures for these goals can be important. Similarly for the Results Chain (OCD 2.1).
- Include a Return-On-Investment (ROI) analysis and breakeven analysis as appropriate. A breakeven analysis takes the form of a graph that indicates the overall costs as surmised from FRD 2.1.1, 2.1.2, 2.1.3, and 2.1.4 (that is starting (up-front) costs plus cumulative costs) subtracted from the value added over time. Clearly indicate the point at which the overall cost is zero. This is where the project begins to show a positive return on the investment. If later cumulative costs potentially bring this return negative later (which is not uncommon, such as when there is an expected downstream cost to purchase a COTS product, upgrades, license fees, implement evolutionary requirements, anticipated domain changes, etc.), then be sure to extend the graph long enough to show this.
- [Consistent with Results Chain (OCD 2.1)]
- [Consistent with Results Chain (OCD 3.2)]
Common Pitfalls:

- Over focusing on costs and underemphasizing the value-added estimate and return on investment.

## 2.2 Requirements Satisfaction

- This section summarizes how well a system developed to the product architecture will satisfy the system requirements.

Common Pitfalls:

- Simply restating the requirements, without showing how and why the proposed architecture guarantees that they will be met.

- Requirements satisfaction is demonstrated by indicating explicitly (usually by model element trace references) why the SSRD requirements satisfy (are true for) the OCD operational concepts and then indicating why the SSAD designs satisfy (implement) the SSRD requirements. If this is done in the SSAD, it should be referenced rather than repeated in the FRD.

### 2.2.1 Operational Concept Satisfaction

- Summarize the product's ability to satisfy the key operational concept elements and critical scenarios, including critical off-nominal scenarios (failover or exception-handling Scenarios).

- Show explicitly why the SSRD requirements satisfy (are true for) the OCD operational concepts. Complete coverage of the System Capabilities (OCD 4.2) is essential. To a lesser degree, Project Goals and Constraints (OCD 4.2) and Levels of Service (OCD 4.4).

- [Consistent with Operational Scenarios (OCD 3.4.3), and all of OCD 4.]

### 2.2.2 Project Requirements Satisfaction

- Summarize how project requirements are being met through the approach adopted for the project and described in LCP 4.

- Explain explicitly (i.e. provide detailed references) how the designs and the plan for implementing the designs are compatible the Project Requirements (SSRD 2). Complete coverage of the Project Requirements is essential. Provide rationale only for complex or high-risk requirements. For most, a simple trace map or matrix from design elements to the requirements should suffice. For those covered in the LCP (e.g. choices of platforms, tools, or programming languages), a reference is sufficient.

- [Consistent with System Requirements (SSRD 2)]

### 2.2.3 Capability Requirements Satisfaction

- Show evidence that the system developed to the product architecture will satisfy the capability requirements, e.g., “capability described/demonstrated/exercised as part of included COTS component”, with a pointer to the results.

- Explain explicitly (i.e. provide detailed references) how the designs will implement the SSRD 3.3 Capability Requirements. Complete coverage of the Capability Requirements is essential. Provide rationale only for critical requirements. For most, a simple trace map or matrix from design elements to the requirements should suffice.
• There is no need to restate obvious mappings from the requirements to the architecture covered in the SSAD or by a requirements traceability tool. But for critical, higher-risk capabilities (e.g., natural language processing, pattern recognition, agent coordination) a full rationale with supporting evidence is needed.

• For each critical requirement, indicate:
  • Criticality: Describe how essential this requirement is to the overall system
  • Technical issues: Describe any design or implementation issues involved in satisfying this requirement. Provide evidence of their resolution via analysis, prototyping, simulation, or technology maturity studies.
  • Cost and schedule: Describe the relative or absolute costs associated with the technical issues associated with satisfying the requirement, and show their consistency with the budgets and schedules in the LCP.
  • Dependencies: Provide evidence of the feasibility of dependencies on COTS package capabilities, externally furnished components, etc.
  • Side effects: For solutions that may involve significant interactions with other requirements such as Levels of Service (see Table 13), provide evidence that the combination of requirements remains feasible.
  • Risks: Describe the circumstances under which this requirement might not be satisfied, and what actions can be taken to reduce the probability of this occurrence.

• [Consistent with System Requirements (SSRD 3.2)]

2.2.4 Interface Requirements Satisfaction

• Show evidence that the system developed to the product architecture will satisfy the critical interface requirements.

• Explain explicitly (i.e., provide detailed references) which designs will implement the SSRD 4 Interface Requirements. Complete coverage of the Interface Requirements is essential. Provide rationale only for critical interfaces. For most, a simple trace map or matrix from design elements to the requirements should suffice.

• Potentially critical or high-risk interfaces include those with external developers; COTS or non-developmental items (NDI); separately evolving interfacing systems; complex user interfaces; and complex hardware interfaces.

• [Consistent with System Interface Requirements (SSRD 4)]

2.2.5 Level of Service Requirements Satisfaction

• Ambitious Level of Service requirements and their tradeoffs can be the most difficult requirements to satisfy, and the most difficult requirements to demonstrate satisfaction of in advance.

• Table 29 summarizes the most effective product and process strategies available for the major Level of Service attributes.

• Table 30 shows how some architecture/product strategies for ensuring a given Level of Service attribute reinforce or conflict with ensuring other attributes.

• Table 31 summarizes the most effective analysis methods available for given attributes and their tradeoffs with other attributes.

• Explain explicitly (through analysis, detailed references to prototypes, models, simulations, etc.) how the designs will satisfy the SSRD L.O.S. Requirements (SSRD 5). Complete coverage of the L.O.S.
Guidelines for MBASE

Feasibility Rationale Description (FRD)

Requirements is essential. Provide rationale only for critical requirements. For many, a simple trace map or matrix from design elements to the requirements should suffice.

- [Consistent with Level of Service Requirements (SSRD 5)]

Common Pitfalls:

- Not explicitly justifying why designs will satisfy L.O.S. Requirements. An unvalidated L.O.S. requirement is a serious project risk.

Table 29: Level of Service Product and Process Strategies

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Product Strategies</th>
<th>Process Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Input/output, Fault-tolerance Functions, Input Acceptability Checking, Integrity Functions, Intrusion Detection &amp; Handling, Layering, Modularity, Monitoring &amp; Control, Redundancy</td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>Error-reducing User Input/output, Help/explanation, Modularity, Navigation, Parametrization, UI Consistency, UI Flexibility, Undo, User-programmability, User-tailoring</td>
<td>Prototyping, Usage Monitoring &amp; Analysis, User Engineering, User Interface Tools, User Involvement</td>
</tr>
<tr>
<td>Performance</td>
<td>Descoping, Domain Architecture-driven, Optimization (Code/ Algorithm), Platform-feature Exploitation</td>
<td>Benchmarking, Modeling, Performance Analysis, Prototyping, Simulation, Tuning, User Involvement</td>
</tr>
<tr>
<td>(Evolvability / Portability)</td>
<td>Descoping, Domain Architecture-driven, Modularity, Reuse</td>
<td>Design To Cost/schedule, Early Error Elimination Tools And Techniques, Personnel/Management, Process Automation, Reuse-oriented Processes, User &amp; Customer Involvement</td>
</tr>
<tr>
<td>Development Cost / Schedule</td>
<td>Descoping, Domain Architecture-driven, Portability Functions</td>
<td>Domain Architecting, Reuser Involvement, Reuse Vector Specification &amp; Verification</td>
</tr>
<tr>
<td>Reusability</td>
<td>Domain Architecture-driven, Portability Functions</td>
<td>Analysis, Continuous Process Improvement, Incentivization, Peer Reviews, Personnel/Management Focus, Planning Focus, Requirement/ design V&amp;V, Review Emphases, Tool Focus, Total Quality Management</td>
</tr>
<tr>
<td>All of Above</td>
<td>Descoping, Domain Architecture-driven, Reuse (For Attributes Possessed By Reusable Assets)</td>
<td></td>
</tr>
</tbody>
</table>
### Table 30: Level of Service Strategies and Relations: Architecture Strategies

<table>
<thead>
<tr>
<th>Primary Attribute</th>
<th>Architecture Strategy</th>
<th>Other Attribute Reinforcement</th>
<th>Other Attribute Conflicts</th>
<th>Special Cases, Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependability</td>
<td>Input acceptability checking</td>
<td>Interoperability, Usability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reinforcement</td>
<td></td>
<td>Development Cost/schedule, Evolvability, Performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Backup/recovery</td>
<td></td>
<td>Development Cost/schedule, Usability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Monitoring &amp; Control</td>
<td></td>
<td>Development Cost/schedule, Performance</td>
<td>Performance reinforcement in long term via tuning</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Input acceptability checking</td>
<td>Dependability, Usability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Layering</td>
<td>Evolvability/Portability, Reusability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>Error-reducing user input/output</td>
<td>Dependability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Input acceptability checking</td>
<td>Dependability, Interoperability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
<tr>
<td>Performance</td>
<td>Architecture balance</td>
<td>Cost/Schedule</td>
<td>Cost/Schedule</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain architecture-driven</td>
<td></td>
<td>Cost/Schedule</td>
<td></td>
</tr>
<tr>
<td>Evolvability/Portability</td>
<td>Layering</td>
<td>Interoperability, Reusability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
<tr>
<td>Cost/Schedule</td>
<td>Architecture balance</td>
<td>Performance</td>
<td>Cost/Schedule</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain architecture-driven</td>
<td>Performance</td>
<td>Cost/Schedule</td>
<td></td>
</tr>
<tr>
<td>Reusability</td>
<td>Domain architecture-driven</td>
<td>Interoperability, Reusability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Layering</td>
<td>Interoperability, Evolvability/Portability</td>
<td>Development Cost/schedule, Performance</td>
<td></td>
</tr>
</tbody>
</table>
### Table 31 Top-Level Field Guide to Software Architecture Attribute Analysis Methods

<table>
<thead>
<tr>
<th>Levels of Service</th>
<th>Methods</th>
<th>Examples</th>
<th>Potential Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Static integrity (partial)</td>
<td>Interface Checking</td>
<td>DOORS, RDD-100, Requisite Pro</td>
<td>• Dynamic integrity</td>
</tr>
<tr>
<td>• Traceability</td>
<td></td>
<td></td>
<td>• Performance, cost, schedule analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Subjective attributes</td>
</tr>
<tr>
<td>• Static, dynamic integrity</td>
<td>Formalized Models</td>
<td>Rapide, Wright, HDM, ACME, Prism</td>
<td>• Model granularity and scalability</td>
</tr>
<tr>
<td>• Security integrity</td>
<td></td>
<td></td>
<td>• Cost, schedule, reliability, full performance</td>
</tr>
<tr>
<td>• Interoperability</td>
<td></td>
<td></td>
<td>• Subjective attributes</td>
</tr>
<tr>
<td>• Subjective attributes:</td>
<td>Scenario Analysis</td>
<td>SAAM</td>
<td>• Largely manual, expertise-dependent</td>
</tr>
<tr>
<td>Usability, modifiability</td>
<td></td>
<td></td>
<td>• Scenario representativeness; method scalability</td>
</tr>
<tr>
<td>• Human-machine system attributes:</td>
<td></td>
<td></td>
<td>• Verification/Validation/Accreditation</td>
</tr>
<tr>
<td>Safety, security, survivability</td>
<td></td>
<td></td>
<td>• Integrity, performance, cost, schedule analysis</td>
</tr>
<tr>
<td>• Performance analysis</td>
<td>Simulation; Execution</td>
<td>Network Simulators; Middleware Instrumentation</td>
<td>• Model granularity and scalability</td>
</tr>
<tr>
<td>• Dynamic integrity</td>
<td></td>
<td></td>
<td>• Input scenario representativeness</td>
</tr>
<tr>
<td>• Reliability, survivability, accuracy</td>
<td></td>
<td></td>
<td>• Verification/Validation/Accreditation</td>
</tr>
<tr>
<td>• Cost, schedule analysis</td>
<td>Parametric Modeling</td>
<td>COCOMO, SMERFS, Queuing Models</td>
<td>• Subjective attributes</td>
</tr>
<tr>
<td>• Reliability, availability analysis</td>
<td></td>
<td></td>
<td>• Static, dynamic integrity</td>
</tr>
<tr>
<td>• Performance analysis</td>
<td></td>
<td></td>
<td>• Verification/Validation/Accreditation</td>
</tr>
<tr>
<td>• Input validation</td>
<td></td>
<td></td>
<td>• Input validation</td>
</tr>
</tbody>
</table>

#### 2.2.6 Evolution Requirements Satisfaction

- Show evidence that the system developed to the product architecture will satisfy the critical evolution requirements (e.g., show which parts of the architecture ensure an easy transition to support of lower-priority post-IOC features; to an upcoming COTS product replacement; to a significant increase in transaction volume; or to a combined operation with another application).

- [Consistent with Evolution Requirements (SSRD 6)]

#### 2.3 Stakeholder Concurrence

- Summarize stakeholder concurrence by reference to:
  - Win–Win negotiation results
3. Process Rationale

This section analyzes the ability of the development to satisfy the stakeholders' (e.g. customer) cost and schedule constraints.

Integration and Dependencies with other components:

Like the previous section, this section is also highly dependent on other documents, foremost the Life Cycle Plan (LCP) and System and Software Requirements Description (SSRD). FRD 3.1 maps primarily to the capabilities in SSRD and milestones in LCP 2.2. FRD 3.2 is a summary of LCP 2.1 and 2.2, with emphasis on priorities above. FRD 3.3 is reasoning that the LCP is consistent and doable (especially LCP 4.).

3.1 System Priorities

- Summarize priorities of desired requirements. Capability and Interface Requirements may be expressed either in priority order or by priority category (e.g., Essential, Important, Desired, Optional or Very High…Very Low). Level of Service Requirements may be prioritized similarly or also by identifying Desired and Acceptable Levels of Service. Project and Evolution Requirements are a mix of Capability and Level of Service Requirements, and should be prioritized as such.

- These priorities should be consistent with the Organization Goals (OCD 3.2) and Project Goals (OCD 4.2) as well as System Requirements (SSRD 4.2).

- Priorities should be distributed somewhat evenly across the priority levels.

Common Pitfalls:

- Making everything an Essential or Very High priority.

- Prioritizing on a capability by capability basis instead of a requirement by requirement basis.

577 Guidelines:

- Use the Easy WinWin prioritization capability for multi-stakeholder prioritization.

3.2 Process Match to System Characteristics and Priorities

- Provide rationale for

  - Choice of process model: The decision table (Error! Reference source not found.) provides guidance on selecting an appropriate process model for various combinations of system objectives, constraints and alternatives.

  - Choice of increment, block, or build sequence in incremental development.
3.3 Consistency of Priorities, Process and Resources

• Provide evidence that priorities, process and resources match:
  • Budgeted cost and schedule are achievable
  • No single person is involved in two or more full-time tasks at any given time
  • Low priority features can be feasibly dropped to meet budget or schedule constraints

• Using the estimated Effort (Person-months) and Schedule from Budgets (LCP 5.2), show that the staffing levels are enough, and that the project is achievable within the schedule.

• It is important to use a credible and repeatable estimation technique for the Effort and the Schedule.

577 Guidelines:

• Given the team projects’ fixed 24-week schedule, use of the Schedule as Independent Variable (SAIV) process is highly recommended:
  a. Use the Easy WinWin prioritization capability to prioritize requirements
  b. Use COCOMO II and expert judgment to determine how many top-priority requirements can be developed with 90% confidence in 24 weeks. Define a useful Core Capability of such top-priority requirements as Increment 1.
  c. Architect the system for ease of adding and dropping borderline-priority features.
  d. Monitor and control progress and introduction of new higher-priority features to keep the Core Capability delivery feasible.
  e. If (usually) the Core Capability is finished early, add next-priority features as the schedule allows.
## Table 32: Process Model Decision Table

<table>
<thead>
<tr>
<th>Objectives, Constraints</th>
<th>Alternatives</th>
<th>Model</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Growth Envelope</td>
<td>Understanding of Requirements</td>
<td>Robustness</td>
<td>Available Technology</td>
</tr>
<tr>
<td>Limited</td>
<td>Limited</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Limited</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Limited to Large</td>
<td>Medium to High</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Limited to Medium</td>
<td>Low</td>
<td>Low-Medium</td>
</tr>
<tr>
<td></td>
<td>Limited to Large</td>
<td>Medium to High</td>
<td>Large Reusable Components</td>
</tr>
<tr>
<td></td>
<td>Very Large</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Medium to Large</td>
<td>Low</td>
<td>Medium</td>
</tr>
</tbody>
</table>

## Table 33: Conditions for Additional Complementary Process Model Options

<table>
<thead>
<tr>
<th>Design-to-cost or Design-to-schedule</th>
<th>Fixed Budget or Schedule Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental Development (only one condition is sufficient)</td>
<td>Fixed Budget or Schedule Available; Early Capability Needed; Limited Staff or Budget Available; Downstream Requirements Poorly Understood; High-Risk System Nucleus; Large to Very Large Application; Required Phasing With System Increments</td>
</tr>
</tbody>
</table>
4. Project Risk Assessment

Any combinations of capabilities or objectives whose feasibility is difficult to assure, are major sources of risk. Risk Assessment consists of risk identification, risk analysis and risk prioritization. Frequent major sources of risk and techniques for resolving them are given in Table 34. The project's overall life cycle strategy described in LCP 2.1 should be consistent with its approach to risk management. The initial set of risks defined here will be updated throughout the project.

- Identify the major sources of risk in the project.
- Provide a description of all identified risks for the project, including risk exposure quantities.
- For critical risks, indicate the following:
  - Description
  - Risk Exposure: Potential Magnitude and Probability of Loss
  - Risk Reduction Leverage: in reducing risk exposure
  - Actions to Mitigate Risk
  - Contingency Plan
  - Identify low-priority requirements that can be left out in the case of schedule slippage
Table 34: Software Risk Management Techniques

<table>
<thead>
<tr>
<th>Source of Risk</th>
<th>Risk Management Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personnel shortfalls</td>
<td>Staffing with top talent; key personnel agreements; team-building; training; tailoring process to skill mix; peer reviews.</td>
</tr>
<tr>
<td>Schedules, budgets, process</td>
<td>Detailed, multi-source cost and schedule estimation; cost/schedule as independent variable; incremental development; software reuse; requirements descoping; adding more budget and schedule; outside reviews.</td>
</tr>
<tr>
<td>COTS, external components</td>
<td>Benchmarking; peer reviews; reference checking; compatibility prototyping and analysis; usability prototyping</td>
</tr>
<tr>
<td>Requirements mismatch</td>
<td>Requirements scrubbing; prototyping; cost-benefit analysis; design to cost; user surveys</td>
</tr>
<tr>
<td>User interface mismatch</td>
<td>Prototyping; scenarios; user characterization (functionality; style, workload); identifying the real users</td>
</tr>
<tr>
<td>Architecture, performance, quality</td>
<td>Simulation; benchmarking; modeling; prototyping; instrumentation; tuning</td>
</tr>
<tr>
<td>Requirements changes</td>
<td>High change threshold: information hiding; incremental development (defer changes to later increments)</td>
</tr>
<tr>
<td>Legacy software</td>
<td>Reengineering; code analysis; interviewing; wrappers; incremental deconstruction</td>
</tr>
<tr>
<td>Externally-performed tasks</td>
<td>Pre-award audits; award-fee contracts; competitive design or prototyping</td>
</tr>
<tr>
<td>Straining computer science</td>
<td>Technical analysis; cost-benefit analysis; prototyping; reference checking</td>
</tr>
</tbody>
</table>

Additional Guidelines:

There are numerous risk identification and analysis tools that can be applied in this section (the COCOMO II Risk feature is useful here). However, they can only give guidelines, not real answers. The best preparation for this section is to try to construct the Feasibility Rationale Description and see where you have difficulties.

Common Pitfalls:

- Simply repeating **Error! Reference source not found.** as your risk analysis
- Repeating your FRD 4 table in the Risk Management section of LCP (LCP 4.1)

5. Analysis Results

- Identify architectural alternatives and tradeoffs. Identify unfeasible architectures or rejected alternatives; document criteria for rejection to avoid having the rejected architectural alternative selected in ignorance at some other point.
- Describe feasible architectural alternatives which were rejected due to solution constraints on the way that the problem must be solved, such as a mandated technology. Those architectural alternatives may be reconsidered should the solution constraints be relaxed.
5.1 Product Features

5.1.1 Advantages

This paragraph shall provide a qualitative and quantitative summary of the advantages to be obtained from the new or modified system with respect to the Organization Goals and Activities. This summary shall include new capabilities, enhanced capabilities, and improved performance, as applicable, and their relationship to deficiencies identified in the Current System Shortfalls, as well as the rationale for new capabilities. For a quantitative analysis, you may reference the Business Case Analysis from the FRD 6.

You may also describe the relationship of this system with any other systems if they exist. Specify if this system is intended to be stand-alone, used as a component in a larger product, or one of a family of products in a product line. If the latter, this section discusses the relationship of this system to the larger product or to the product line.

5.1.2 Limitations

This paragraph shall provide a qualitative and quantitative summary of potential disadvantages or limitations of the new or modified system. These disadvantages and limitations shall include, as applicable, degraded or missing capabilities, degraded or less-than-desired performance, greater-than-desired use of computer hardware resources, undesirable operational impacts, conflicts with user assumptions, and other constraints. These are used either for stakeholder expectations management or as a basis for further negotiation of system capabilities or tradeoffs.

5.1.3 Tradeoffs Considered

This paragraph shall identify and describe major alternatives for the concept of operation of the system, their characteristics, the tradeoffs among them, and rationale for the decisions reached. Also discuss alternative architectures and their pros and cons.

5.1.4 Changes Considered

- These are changes considered but not included.
- In general, the results of the Win–Win requirements negotiation activity will be to drop or defer some capabilities from the initially proposed system. It is valuable to capture these for future reference, along with the rationale for dropping or deferring them. Some of those changes considered but not included may become Evolution Requirements.
- Include Reference to Win–Win artifact (if applicable)
- You may include a threshold for including some of the deferred capabilities (e.g., depending on the availability of a specific COTS package, etc.)
- [Consistent with Evolution Requirements (SSRD 6)]

5.2 Commercial-Off-The-Shelf Solutions

- List of existing COTS products that should be investigated as potential solutions
- Reference any surveys or evaluations that have been done on these products
- Is it possible to buy something that already exists or is about to become available? It may not be possible at this stage to say with a lot of confidence, but any likely products should be listed here.
- Consider whether there are products that must not be used, and state the reason.
6. Appendices

- Provide details of cash flow and project earnings statement as appropriate.
Guidelines for MBASE

Construction, Transition, & Support (CTS)

A. General Construction Process Guidelines

- The process for the Construction and Transition Phase should be risk-driven: in particular, you should follow process strategies, to accommodate your particular requirements. For instance, if you have stringent performance requirements, you should plan accordingly for some of the following process strategies, such as Benchmarking, Modeling, Performance Analysis, Prototyping, Simulation, Code Profiling, Tuning and Optimization (Reference Table 8 in Hoh In’s Dissertation).

- It is critical to keep all the artifacts properly baselined. In particular, at the end of each iteration, the Operational Concept Description (OCD), System and Software Requirements Definition (SSRD), System and Software Architecture Description (SSAD), Life Cycle Plan (LCP), Feasibility Rationale Description (FRD) must be consistent with the IOC plans and implementation (e.g. source code comments, component and object names, etc.) documentation. This is consistent with the concept of “continuous integration” aspect of iterative and incremental application development.

- As part of making winners of all the success critical stakeholders, it is recommended that your clients assess and evaluate each one of the intermediary or incremental releases, to avoid any changes introduced late in the process, which might introduce schedule slippage--something you want to avoid in a design-to-schedule situation.

- Reference information where applicable, as opposed to repeating information. In particular, you should provide references to the information, when the reader will have to look at multiple documents to get hold of the information. In particular, it is recommended to use hyperlinks with traceability matrices that reference specific related areas in the documentation.

- Although the purpose of concurrent engineering is to having coding and testing proceeding in parallel, it is advisable to have a functional freeze at some point during the iteration. Ideally, the release at the end of the iteration should have thoroughly tested the features implemented in that increment. If the programmers don't stop adding features at some point before the end of the iteration, the added features may not be thoroughly tested, and furthermore, may compromise the quality or correctness of the current feature set, leading to an unusable increment.

577b Guidelines:

- Rose Model Files should also be kept up-to-date. The code generation as well as the round-trip engineering capabilities of Rose (Java, C++, ...) should be used, where applicable.

- Ideally, during the rebaselining of the LCA packages, you should set your own dates for performing peer reviews: make sure that you turn in the required deliverables by the date indicated on the class schedule.

- Make sure that your project plan also identifies which increments are important to be evaluated by the customer and the users. It is important that the client periodically reviews the software as it is being developed, in particular, regarding user interface considerations. It is very important, due to the short schedule, to minimize rework, by avoiding making assumptions: when in doubt, refer to your customer. We recommend that most teams adopt the practice of delivering intermediate working increments to the clients, and keep incorporating the feedback.
During Construction, you will be performing the following activities:

- Requirements Management
- Detailed Design
- Coding
- Unit and Integration Testing
- Peer Reviews
- Configuration Management
- Quality Management

You will be generating the following artifacts:

- Iteration Plan
- Iteration Assessment Report
- Release Descriptions and Notes
- Test Plans and Results
- Peer Review Plans and Reports
- Quality Management Plans

**Requirements Management**

Changes in the requirements will be documented, as appropriate, in the Operational Concept Description, and the System and Software Requirements. Subsequently this may affect the architecture in the SSAD, impact the schedule and appear in the LCP, for which the FRD will then need to be updated. For instance, some of the changes might be moved to the Changes Considered but not Included (FRD B.5.1.4) or Evolutionary Requirements (SSRD 6.). Accordingly, the Feasibility Rationale Description should be updated to reflect the impacts of the changes on the feasibility criterion of the project (e.g. is the value of the revised system greater than the cost?).

**Design Creating or Modification**

During Construction, a lot of effort will be spent on developing, detailing or changing the system design. The design activities are reflected in the System and Software Architecture Description, and the associated model files (e.g., the Rational Rose model). Low-level design details should also be included as comments in the source code and be consistent with the SSAD (especially naming). Another related activity would be to review the design and implementation (includes design meetings and consultations, as well as formal and informal reviews, walkthroughs, and inspections) and generating accordingly Inspection Reports.

Since there is no separate Detailed Design artifact, the Detailed Design information is documented in the following 3 artifacts:

1. SSAD
2. Rose MDL Files
3. Source Code (comments always related back to SSAD)
577 Guidelines:

You should try to strike a good balance as to what goes in the SSAD, vs what goes in the MDL files, vs what goes in the source code. Because having many objects/operations without tool support can lead to an unwieldy SSAD, you may want to leave very detailed information (e.g., method argument types, return values, ...) in the Rose MDL file. Once you have the information in the Rose model file, you can generate a report out of it (e.g., using SoDA), and include that information in the SSAD, as Operation Specification Templates, and so forth.

At any point in time, you should make sure that there are no glaring problems, such as having your architecture/design as represented in the MDL file conflict what is represented in the SSAD (e.g., block diagram), or with how the system was built.

Code Generation/Modification

During construction, most of the effort will be spent on actually coding the system. During coding, care should be taken to follow proper coding standards and programming style, emphasizing on code readability and maintainability, including the proper use of comments in the source code. An associated activity will consist of code reviews or inspections, where you will be assessing the code for defects, and generating Peer Review Reports. WARNING: risk managed scheduling and resource allocation as well as careful and regular assessment and control are essential for a successful outcome.

Some related activities during the Construction stage will include creating or modifying prototypes, assessing various Commercial Off The Shelf (COTS) components for the application, tailoring COTS products, and integrating COTS products into application including glue code design, development and test.

Testing

Testing is an integral part of the Construction stage. This includes testing individual components of the system, writing test drivers, simulations, and gages, generating regression test packages, writing test descriptions, matching with requirements scenarios and reporting test results.

Project Management and Special Functions

Throughout the project, you will be performing planning and control activities, such as creating or modifying plans, reporting status, collecting and analyzing metrics, managing or coordinating work (configuration management, quality control). In particular, the Project Manager will be generating Iteration Plans and Iteration Assessment Reports.

Configuration Management and Quality Management: Hours spent performing configuration management and quality management functions, including developing Quality Management Plan, Peer Review Plan, coordinating tools and the like must be planned and accounted for on the construction schedule.

In preparation for, and during the Transition Phase, you will be performing several activities, such as developing and executing the transition plan, coordinating deliverables with the client, meeting with key personnel for transition strategy and readiness discussions. You will also be training the users on the application, developing training material, as well as developing user documentation (e.g., user's manual and online help). Finally, you will need to spend some time coordinating, preparing and packaging customer deliverables for delivery (source code files, installation scripts, maintenance package, regression test package, support tools and environment, etc.). Most importantly documenting how the system must be supported and will support anticipated evolutionary changes.

B. Guidelines for the Deliverables

The artifacts of the process grouped in "logical" sets. These groupings do not imply physical document groupings. They indicate what areas are the main focus within a particular phase.

Requirements, Architecture, Design and Management Set
Guidelines for MBASE Construction Transition & Support (CTS)

- Operational Concept Description (OCD)
- System and Software Requirements Definition (SSRD)
- System and Software Architecture Description (SSAD) and Rose Model Files (MDL)
- Feasibility Rationale Description (FRD)
- Life Cycle Plan (LCP). Must include effort/cost estimates such as:
  - COCOMO II run
  - COCOMO II Data Collection Form (as an Appendix) as a rationale capture for the COCOMO Estimate
- Risk-Driven Prototype(s)

Construction Planning Set

- Life Cycle Plan (LCP)
- Quality Management Plan (including guidelines for Configuration Management, Testing and Peer Reviews)
- Peer Review Plan
- Test Plan

Status Assessment Set

- Weekly Effort Forms
- Weekly Status Reports

Construction Working Set

One Construction Set is delivered at the end of each iteration.

- Documentation
  - As-built specs
    - As-built Operational Concept Description (OCD)
    - As-built System and Software Requirements Definition (SSRD)
    - As-built System and Software Architecture Description (SSAD)
    - As-built Rose Model Files (MDL)
    - As-built Feasibility Rationale Description (FRD)
    - Updated Risk Management Plans
    - Summary of the revisions
Guidelines for MBASE

• Iteration Plans (one per iteration)
• Peer Review Reports (at least 1 peer review report per iteration)
• Test Reports (at least 1 test report per iteration)
• Release Description (one per iteration)
• Iteration Assessment Reports (one per iteration)

• Implementation
  • Source Code Baselines (including comments in the source files and “Read Me” files)
  • Associated Compile-Time Files
  • Component Executables
  • Test drivers and simulations

Transition Set

• Transition Plan (including some Training planning)
• User Manual
• Transition readiness assessment

Support Set

• Support Plan (including evolution support plan)
• Training materials (including tutorials and sample data)
• Regression Test Package
• Packaged Tools and Procedures

Data Collection Set

• Size Report (including Source Lines Of Code (SLOC) estimates) *(one size report per iteration)*
• Other data collection items such as:
  • COCOMO Data Collection Form (including actuals)
  • COCOTS Data Collection Form

577b Guidelines:

All deliverables should be properly stored in the CS 577 Archive, in accordance with the course guidelines. Below is a sample CS 577b start of construction schedule:
C. General Guidelines for Plans and Reports

The following guidelines for plans and reports are very general. As such some items and activities they describe may not be apply to the particular project at hand. In some cases items will need to be added. The choice as to what to include and at what level of detail should be risk driven in accordance with achieving high system assurance and effective development team communication given the constraints of the project. Below are some questions that may be helpful in determining the appropriate items to include and their respective level of detail:

What should be documented to indicate that the correct system would be constructed?

What should be documented to indicate that the system would be constructed correctly?

Are the plans and processes helping to guide and control the construction or do they hinder it?

Are all the development team members being utilized effectively?

Do the plans address all the significant construction issues (in accordance with the FRD feasibility analysis)?

Try to keep plans and reports short, tightly focused, and as concise as possible. Keep in mind that the audience is generally developers who are likely already familiar with the system concept and as such extended explanations, justifications, and so forth are unnecessary. Be direct, brief, and clear as possible about what is asked for or being reported. Consider the use of tables, diagrams, bullet lists and so forth over large blocks of text.

The following table presented within the “High-Level Dependencies” will indicate the general level of integration a particular plan or report has with LCO/LCA MBASE deliverables:

<table>
<thead>
<tr>
<th>OCD</th>
<th>SSRD</th>
<th>SSAD</th>
<th>LCP</th>
<th>FRD</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Where X is one of:
Guidelines for MBASE

√ = direct integration
+
= strong influence
~
= moderate integration
- = indirect integration
Iteration Plan

A. Description

1. Purpose

Overall the purpose of the iteration plans are to detail the incremental implementation and control of the SSRD requirements following the designs within the SSAD according to the schedule and approach specified in the LCP. The Iteration Plan for an upcoming iteration is planned in the current iteration. It is modified as needed during the iteration. The current iteration plan is an input to the next iteration plan. There are often two such plans: one for the current iteration, and one under construction for the next iteration. An iteration plan is realized and frozen after the scheduled iteration time is exhausted. The next iteration plan is then executed and an iteration assessment report is generated for the previous iteration. The Iteration Plan corresponds to the ‘Establish next level objectives, constraints and alternatives’ in the Win–Win spiral model.

2. Timing

3. Intended Audience

The purpose of the Iteration Plan is to keep the construction on track and focused on realizing the SSAD designs. All the project stakeholders should be familiar with the Iteration Plan, and in particular, the development team.

4. Participants

5. Responsibility

The Project Manager is responsible for authoring the Iteration Plan and keeping it up-to-date.

6. Completion Criteria

7. Additional Information

577b Guidelines:

The following figure is a sample schedule for iteration activities within 577b.
8. High-Level Dependencies

The Iteration Plan requires the following as inputs:

- Life Cycle Plan for the overall milestones to be achieved for each iteration (i.e. schedule estimates, dependencies, etc.)

- Life Cycle Plan and Feasibility Rationale for the identification and assessment of the risks and the risk management strategy to be implemented during each iteration

- System and Software Requirements Definition (SSRD) for the list of requirements that must be completed

- Current status of the project (as represented by the set of Weekly Status Reports) to-do’s, unrealized tasks from previous iterations.

- Current Test Reports and Peer Reviews Reports for a summary of the defects that must be removed prior to next release.

```
<table>
<thead>
<tr>
<th>OCD</th>
<th>SSRD</th>
<th>SSAD</th>
<th>LCP</th>
<th>FRD</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>+</td>
<td>+</td>
<td>√</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
```

B. Document Sections

1. Iteration Overview

Provide a high-level overview of the content of the given iteration. Indicate which LCP milestones will be addressed.

1.1 Capabilities to be Implemented

- Identify the features, requirements or use-cases that are being developed (implemented, tested, ...) for this iteration.

- Provide reference to particular System Capabilities (OCD 3.), System Requirement (SSRD 3.2), Level of Service Requirement (SSRD 5.)
• Provide also reference to the various artifacts (non-source code) that will be developed during the iteration. E.g. COTS configuration

Additional Guidelines

Each component should be accounted for in at least one iteration. All requirements should be implemented and tested (or re-negotiated) by the completion of all the iterations. Be mindful of implementation dependencies. Document complex dependencies and communicate them to the appropriate development staff.

1.2 Capabilities to be Tested

• Identify the software features and combinations of software features to be tested this iteration. This may also include non-functional requirements or extra-functional requirements, such as performance, portability, and so forth.

Every requirement listed in the SSRD LCA package, should be tested:

• Project requirements (SSRD 2.)
• System requirements (SSRD 3)
• Interface requirements (SSRD 4.)
• Level of Service requirements (SSRD 5.)
• Evolutionary requirements (SSRD 6.)

Additionally you may need to test non-requirement component features such as COTS capabilities and quality, API functionality, etc.

1.3 Capabilities not to be tested

Identify notable features, and significant combinations of features, which will not be tested this iteration and why (e.g. a given feature uses a feature which will be implemented in following iteration).

1.4 Objectives

State measurable goals to be achieved during the iteration in terms of items such as:

• Implemented use-cases
• Defect density
• Successfully executed test cases
• Risks Addressed
• Performance Levels
• Functionality
• Capacity
Describe specific, measurable, achievable, relevant and time-limited ("SMART") objectives that can be demonstrated (for instance within an in Iteration Assessment Report, or to a Review Board) with this iteration. It is acceptable to specify both desirable as well as acceptable levels. The time-limited aspect should be emphasized to ensure the objectives are realized, e.g. "at least ten test cases will be executed per day."

2. Plan

This is the core section of the Iteration Plan. It is important to keep the Plan up-to-date during a given iteration. Thus, this section should be written so that it is very easily modified and updated. Be sure to keep careful version control.

2.1 Tool Support

We recommend the use of a project management tool, such as Microsoft Project, which provides ways of describing timelines, milestones and resources.

2.2 Schedule of Activities

Provide detailed diagrams showing timelines, intermediate milestones, when testing starts, beta version, demos etc. for the iteration. This should detail major milestones indicated on the lifecycle schedule within LCP x.y

2.3 Resources

- Describe the Resources needed for this iteration – human, financial, etc.
- Highlight the resources that are on the critical path.
- Describe constraints or dependencies on resources during this iteration.

2.4 Team Responsibilities

- Provide detailed team responsibilities, covering the possible range of activities for this particular iteration.

3. Approach

Describe the general approach to be followed for this iteration. Note any special need, constraints, or opportunities to be leveraged during this iteration. Provide references to specific relevant items within various plans (do not repeat them here):

- Risk Management plans (e.g., from LCP x.y and FRD x.y)
- Quality Management Plan from CTS identifying quality management strategies
- Test Plan identifying the test strategy for this particular iteration.

4. Assumptions

Describe briefly the specific (significant) assumptions, under which this plan will hold: i.e., if those assumptions were no longer satisfied, the Iteration Plan would have to be revisited.
Iteration Assessment Report

A. Description

1. Purpose

An iteration is concluded by an iteration assessment, where the actual results of construction actively are assessed in the light of the evaluation criteria that were established within the iteration plan. Iteration Assessments are not updated, but should be maintained for future reference. One aspect of the Iteration Assessment Report is to come up with "Lessons Learned", which corresponds to the ‘Evaluate Product and Process alternatives’ in the Win–Win Spiral Model.

2. Timing

3. Intended Audience

4. Participants

5. Responsibility

The Project Manager is responsible for the Iteration Assessment. However, all the development team contributes to the content of the Iteration Assessment Report.

6. Completion Criteria

7. Additional Information

This assessment is a critical step in an iteration and should not be skipped. If iteration assessment is not done properly, many of the benefits of an iterative approach will be lost. Note that sometimes the right thing to do in this step is to revise the evaluation criteria rather than reworking the system. Sometimes the benefit of the iteration is in revealing that a particular requirement is not important, too expensive to implement, or creates an unmaintainable architecture. In these cases, a cost/benefit analysis must be done and a business decision must be made. Sound metrics should be used as the basis of this assessment.

8. High-Level Dependencies
B. Document Sections

1. Overview

1.1 Capabilities Implemented

- List the features, use-cases and scenarios (from OCD x.y), and their respective requirements (SSRD x.y), components and objects that were actually implemented.
- Indicate divergence from items planned to be implemented within 1.1 of the Iteration Plan.

1.2 Summary of Test Results

- Provide an overall assessment of the system as demonstrated by the test results
- Summarize the evaluation of the test items
- Report any variances of the test items from their design specifications
- Indicate any variances from the test plan, test designs, or test procedures. Specify the reason for each variance.
- Evaluate the comprehensiveness of the testing process against the comprehensiveness criteria specified in the test plan if the plan exists.
- Identify the features or combinations which were not sufficiently tested and highlight those as needing further testing in the following iteration
- Identify all resolved incidents and summarize their resolutions
- Identify all unresolved incidents.

1.3 Open Problems

- Identify any remaining (open) deficiencies, limitations, or constraints that were detected by the testing performed. Problem/change reports may be used to provide deficiency information. For each remaining (open) deficiency, limitation, or constraint, describe the following:
  - Its impact on system performance, including identification of requirements not met
  - The impact on system design to correct it
  - A recommended solution/approach for correcting it

1.4 Objectives Reached

- Assess the results of the iteration relative to the evaluation criteria that were established for 1.4 Objectives within the Iteration Plan.
2. Adherence to Plan

Describe how well the iteration ran according to plan. Was it on budget and on time? Provide some insight to avoid mistakes for future iterations.

3. Approach Used and Suggested Changes

With respect to Iteration Plan 3.0:

- Summarize the major activities and events: resource consumption, total staffing level, …
- Evaluate any improvements in the approach that should be incorporated into the following iteration.
- Provide suggestions for improvements to the environment: tools, resources, etc…

4. External Changes Occurred

- Describe any changes that have occurred with respect to the original assumptions in Iteration Plan 4.0: e.g., changes in requirements, new user needs, competitor’s plan, discovery of a more efficient algorithm, …
- Provide indications on the amount of rework required for the following iteration

5. Suggested Actions

- State any actions suggested due to unexpected results of the analysis.
- Provide any recommended improvements in the design, operation, or testing of the system tested. For each recommendation, describe the impact on the system.
Release Description

A. Description

1. Purpose

The purpose of the Release Description is to describe items, particularly executables that will be made available after the completion of a development incremental plan. A Release Description is prepared right at the end of an iteration. In particular, the final Release Description before the Product Release is the most critical one, as it details the system outcome and aids in transition. Release descriptions are good candidates for “Read Me” files.

2. Timing

3. Intended Audience

The intended audience of a Release Description consists of the development team, as well as the customers. Since it is recommended to have the users and customer evaluate each one of the releases, the Release Description will serve to manage their expectations and smooth the evaluation by noting and anticipating possible problem areas or missed expectations (e.g. don’t use feature X as it presently non functional)

4. Participants

The developers, the testers and the quality management workers, contribute to the Release Description.

5. Responsibility

6. Completion Criteria

7. Additional Information

577b Guidelines:

8. High-Level Dependencies

<table>
<thead>
<tr>
<th>OCD</th>
<th>SSRD</th>
<th>SSAD</th>
<th>LCP</th>
<th>FRD</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>~</td>
<td>+</td>
<td>~</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
B. Document Sections

1. About This Release

Provide version information, what the release consists of, documentation, licensing, etc. Include the following information, as applicable.

1.1 Physical Inventory of materials released

List all the physical media, documentation, and hardware that make up the software version being released by identifying numbers, titles, abbreviations, dates, version numbers, and release numbers as applicable. Include privacy considerations and restrictions regarding duplication and license provisions.

1.2 Inventory of software contents

- List all computer files that make up the software version being released by identifying numbers, titles, abbreviations, dates, version numbers, and release numbers as applicable.
- List the number, title, revision and date of all documents pertinent to this software. This should include applicable requirements, SSRD, design, SSAD, test (CDC Test Description and Results) and user documents.

2. Compatibility Notes

- Describe software (include version) that interact with the system and that are known to be compatible.
- Describe significant software that is known to be incompatible, and if there are any workarounds.

3. Upgrading

- Describe installation, data conversion from information produced by earlier versions, etc.

4. New Features and Important Changes

- Provide an accounting of the differences between this release and the previous (or what there is if this is the first release) for the following areas:

4.1 New Features

- List all the new features incorporated into the software since the previous version.

4.2 Changes since previous release

- List all changes incorporated into the software since the previous version.
- Identify as applicable the problem reports, peer review reports, test results and change notices for each change.

4.3 Upcoming Changes

- List all new features and changes that will be incorporated in future releases.
5. Known Bugs and Limitations

- Identify any possible problems or known errors with the software at the time of release, and instructions for recognizing, avoiding, correcting or handling each error (if known).

6. Defect and Change Request Reporting Procedures

- Provide information for reporting change requests, problems and defects with the current version
- Provide a point of contact to be consulted if there are problems or questions

7. Appendix

Use the appendix for information that does not fit directly in the body of the document.
Quality Management Plan (QMP)

A. Description

1. Purpose

The objective of construction is to follow a high quality process and deliver high quality products. Quality is elusive and poses particularly challenging issues when addressed only after the majority of a system has been implemented. As such it is difficult to achieve directly and a sound set of guidelines established prior to and followed during implementation can help achieve this indirectly and incrementally. It is also difficult to ensure that quality will be achieved as a matter of course, however the main concern is to avoid unspecified, haphazard or ad-hoc and often clashing approaches.

This plan should identify the methods and techniques to be used by the project for "assessing" and tracking the quality of the product being produced during the Construction phase. It needs to include the methods and techniques applicability and timing. The methods and techniques include peer reviews (using the general, Section 2.3 definition of the term); V&V activities done by team members or others (IV&V) and their problem reporting process; problem report tracking; and testing. For areas like testing and peer reviews which get their own plans and reports on a risk driven basis, it serves as a overview, and allows relating them to other areas which do not have separate plans. For other areas like IV&V and problem reporting and tracking it provides all the necessary information.

2. Timing

3. Intended Audience

Developers as well as the maintainers of the software system would use this plan the Quality Focal Point can also use the plan to assess the overall quality approaches of the project.

4. Participants

5. Responsibility

The project manager should ensure that a plan is created, he/she may delegate it to the Quality Focal Point to prepare the plan.

6. Quality Focal Point

It is the responsibility of the Quality Focal Point to implement the plan so that each team member can carry out their designated tasks for achieving required quality without significant additional efforts. The Quality Focal Point performs the quality management activities for the project and is responsible for the overall quality of the product.

577 Guidelines:

For versions of the Quality Management Plan since the last ARB, include a summary of changes made in the document to ease the review process.

577 Guidelines:

It is sufficient to identify the Quality Focal Point for the project and his/her associated responsibilities.
7. Completion Criteria

8. Additional Information

577b Guidelines:

For most projects, the Quality Management Plan would be dictated by Organizational quality assurance mechanisms.

For CS 577b, this document should help the members of the project team understand each person’s contribution to quality. Each team has the flexibility to choose their own standards for quality management and should take the initiative in defining additional quality mechanisms as dictated by project requirements. For instance it should include the related trade offs as the degree of testing decreases and reviews increases, the methodology used to control injection and removal of defects etc. It should be clear from the plan who is primarily responsible for the implementation of the various parts of the plan.

9. High-Level Dependencies

• The Quality Management Plan has the following dependencies within other sections:

<table>
<thead>
<tr>
<th>OCD</th>
<th>SSRD</th>
<th>SSAD</th>
<th>LCP</th>
<th>FRD</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>√</td>
<td>+</td>
<td>~</td>
<td>~</td>
<td>-</td>
</tr>
</tbody>
</table>

B. Document Sections

1. Purpose

1.1 Overview

This section should describe the purpose, scope and audience of this plan. Quality of software is the degree to which software components meet specified requirements (SSRD x.y) and user/customer operational concept (OCD x.y) expectations as well as provide clear value (FRD x.y) with respect to the cost and effort. The purpose of this plan is to ensure that there is adequate direction to achieve the goal of delivering quality software.

Common Pitfalls:

Simply repeating the purpose of the document from the guidelines

1.2 References

This document should refer to external quality management documents (such as process guidelines). References made to all documents along with their versions and numbers should be identified in this section.

577 Guidelines:

A "complete citation" for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]

It should also include a list with the associated links to the process guidelines and coding conventions used in the project.
1.3 Change Summary

For each version of the Quality Management Plan, describe the main changes since the previous version. The goal is to help a reviewer focus on the most critical parts needing review.

577 Guidelines:

For versions of the Quality Management Plan since the last ARB, include a summary of changes made in the document to ease the review process.

2. Quality Guidelines

This section describes the guidelines for quality management. This section should be very brief and only cover those quality tasks that are significant and meaningful to the project.

2.1 Design Guidelines

Briefly describe design guidelines to improve or maintain modularity, reuse and maintenance, etc. In particular indicate how the designs in SSAD x.y will map to the implementation of those designs (e.g. how will the object models be translated into code)

2.2 Coding Guidelines

A team would probably choose to implement various components of its software system in various programming languages. It is necessary though to follow the same spirit of documentation and coding throughout the system for ease of understanding among all the team members and to ensure smooth transition to maintenance. The approach used should attempt to document the code in such a way that it could easily be communicated to an outside developer or new team member that understood the guidelines.

We are providing links to some industry conventions for coding in some of the implementation languages:

- C: http://www.gnu.org/prep/standards_toc.html
- C++: http://www.nfra.nl/~seg/cppStdDoc.html
- Java: http://www.infospheres.caltech.edu/resources/code_standards/java_standard.html


It is not important which coding convention is chosen, but it is very important that the project identifies and complies with defined coding standards (for each language used). You can develop your own coding conventions and provide them in the appendix of this document.

There should also be sufficient comments in the code to support maintenance of the software for instance the change history and modification details. Each module, class, interface and function should be described and related to the SSAD x.y designs.

The header of each source code file should contain the following information. This header should not replace in line code comments (at functions, methods, etc.) that explain non-trivial implementation points.

Version Control and History
Provide a chronological log of the changes introduced to this unit.

**Implementation Considerations**

Provide detailed design and implementation for as-built considerations. A description of how well this code implements an SSAD design (for example object defining qualities are a useful start for this). The SSAD design document may have to be updated to reflect any discrepancies.

**Unit Verification**

Provide links or references to any of the following, as applicable:

- Unit/integrated test descriptions
- Unit/integrated test results
- Code walkthrough/review/inspection results

**Integration**

How the component(s) this code implements fit within the application together with the tests used to verify each version

**Additional Information**

Include any other information that could be useful in understanding the software element.

The main purpose of the comments in the source code is to provide a trail of implementation decisions that aren't documented elsewhere. Ideally, most of the detailed design should be documented in the SSAD. However, this often isn't practical and hence in contrast to external documentation, internal documentation is found within the program listing itself. It's the most detailed kind of information, at the source-statement level. Because it's most closely associated with the code, internal documentation is also the kind of documentation most likely to remain current and correct as the code is modified. This often helps make code easier to maintain and evolve for present and future developers of the system.

Design and coding standards should promote self-documenting artifacts. Having source files in a single, homogeneous format consistent and integrated with the SSAD, will avoid having separate documents that inevitably diverge from the implementation.

Each source file should contain where further elaboration beyond the SSAD and SSRD (due to implementation issues) the following as applicable:

- Developer Updates & Revisions
- Project Updates & Revisions
- Basic Assumptions & Definitions

Having well-commented source files goes a long way towards improving the maintainability of the delivered systems.

### 2.3 Quality Assessment Methods

Quality Assessment methods refer to all the methods, techniques, tools, processes, etc. that [can] identify (or measure) "problems". Some are used to detect the problems earlier than testing: they generate problem reports, issues lists and/or defect lists. These "measures" of the quality of the software system provide information that can be used in closed loop control of the development of the system by accounting for those that have been repaired or
are still to be repaired. Most of the quality assessment methods and defect/problem reporting systems would rapidly spiral out of control without an accompanying configuration management approach, as described in section 2.4.

COQUALMO, the COstructive QUALity parametric MOdel, identifies three major ways to find, and eventually remove, "problems" in the software system: "Automated Analysis, People Reviews and Execution Testing and Tools".

Automated Analysis encompasses "Simple compiler extensions for static module-level code analysis, syntax, type-checking" at the lowest level in the model up to "Formal specification and verification; [and] advanced distributed processing and temporal analysis, model checking, symbolic execution" at the highest level.

People Reviews, also often called "Peer Reviews", encompass "Ad-hoc informal walkthroughs; [with] minimal preparation and/or follow-up" at the low end extending upwards via levels that include

- "processes with well-defined sequence of preparation, review, follow-up;
- ... plus formal review roles and procedures applied to detailed design and code reviews;
- ... plus formal review roles and procedures applied to specification, design code, test, documentation artifacts; [using] basic review checklists, [and] root cause analysis;
- ... plus formal review roles and procedures for fixes, change control, [and] extensive review checklists [and] root cause analysis, [including] continuous review process improvement"

Execution Testing and Tools covers "Ad-hoc testing and debugging, [and] basic text-based debugger" up through "Highly advanced tools for test oracles, distributed monitoring and analysis, assertion checking; Integration of automated analysis and test tools; [and] model-based test process management".

"Defects" are related to changes that must be made in the software system or documentation. Some defects are identified using methods with objective criteria, such as deviations from formal specifications like exit criteria. All other defects can only be defined by using the rule "any change that is made". Since not all defects are equal, especially in evolutionary and/or cyclic approaches to specifying, designing and implementing software, those changes are further categorized as "avoidable" or "unavoidable", and may have different levels or degrees of reporting depending on project and organizational circumstances.

"Issues" are usually those things identified as a result of activities independent of the author of the artifact in which they reside, such as testing. As such, some form of formally identifying them and tracking them is desired so they do not "fall through the cracks" in a rapidly moving processes. Issues are those problems and/or defects which are NOT the responsibility of the "author" of the particular artifact being assessed: they require the attention of and correction by someone else in some other artifact.

Unavoidable defects are those changes (or objectively identified with the author's concurrence) which often arise because of the methods, techniques or approaches being followed that necessitate changes. Examples include changes arising because of

- the dynamics of learning,
- exploration in IKIWISI situations,
- code or screen contents reorganizations taken on as an "afterthought",
- replacement of stubs or placeholders in code, etc.

Such situations are as a result of life cycle and project dynamics.

Avoidable defects are those changes in analysis, design, code or documentation arising from human error, and which could be avoided through better analysis, design, training, etc. Examples include code stub replacements that violate
win conditions or requirements such as execution time or memory space, for instance the replacement of a "stub" which breaks a critical timing constraint.

Three methods of early (before testing) identification of defects are Peer Reviews, Independent Verification and Validation, and Automated Analysis.

2.3.1 Peer Reviews

Quality "assessment" methods, generically called peer reviews since performed by peers in the development team. Cover range from full-up Fagan's inspections to simple buddy checks.

577b Guidelines:

Identify which techniques to be used when/where: some form of multi-person review for detailed design (SSAD and Rose models) and code for "critical" (with respect to capabilities to be provided) components or modules. Identification of any other techniques to be used for non-critical components or modules, such as PSP Personal Reviews, desk checks, informal colleague walkthroughs, or Buddy Checks (Agile Internal Review with two people—the author and a reviewer) or checklists. Other techniques that might be applied to the critical components or modules include Fagan's Inspections or variants thereof, team structure walkthroughs or Agile informal/internal reviews.

The Agile forms are used to capture this data with minimal effort. Also the students track this information in the Weekly Effort Report.

2.3.1.1 Types of peer reviews

- List the types to be used; identify by any reasonable type of reference the definition of the method or technique.
- Identify the kinds and circumstances of the "problems" directly identified by the peers: issues or defects.

2.3.1.2 Degree of data gathering

- At the very least describe the defect and its classification as to severity and avoidability.
  - Defects which are unavoidable and are identified and repaired within an iteration do not have to be reported.
  - Avoidable defects, even if immediately repaired (no problem report generated) must be reported.
- At the very best: gather data to show effectiveness of the assessment techniques AND sufficient defect detail to enable Quality Improvement such as causal analysis leading to specific techniques to prevent defects from being made.

577b Guidelines:

- Specify the extent of the data to be gathered.
- Data gathering forms to be included in an appendix.
- Specify the links to the Quality Reports.

As a part of course work only: It is public to see the defects to aid the team members as well as the researchers in their research.
2.4 Process Assurance

Details of this section for CS577 are under development.

2.5 Product Assurance

2.5.1 Requirement Verification

- Include a Requirements Verification Matrix specifying how each requirement from SSRD will be verified either by Testing, Demonstration, Analysis, Simulation, Inspection, etc.

2.5.2 Independent Verification and Validation

Barry Boehm, in [Boehm '81], identifies various activities in Figure 24-6. Those that are performed throughout the development process include:

- Tool enhancement and maintenance (11%)
- Management and reporting (24%)

Other activities are tied to a Waterfall Phase

- Requirements Analysis (15%)
- Equations Analysis (12%) [which today might also include "Architectural" Analysis]
- Code Analysis (18%)
- Testing (20%)

577b Guidelines:

IV&V to generate independent feedback (either defect lists and/or problem reports). IV&V to concur on classification of defects found by multi-person peer reviews and also needs to review and test all deliverables for completeness and correctness and ensure the verification of the project’s compliance with its documentation and code standards and feasibility of the solution.

2.5.3 Automated Analysis

Identify those techniques and tools which are used in the project.

2.6 Problem Reporting and Tracking System

This section should state the methods to be used for Problem, Issue and Defect reporting and tracking. It is needed to round out the quality assessment "big picture". It should refer forward to the detailed description in section 2.6.

- Identify the system for receiving reports from the field or IV&Vers prior to fielding.
- Identify how it will be monitored and by whom, including frequency of analysis, and weekly summary report generation.
577b Guidelines:

Use of Free Software Foundation systems "gnats"

Any issues at an anchor point needs to be closed. If an issue is deferred or open at an anchor point then its needs to be reported as a problem report summary.

2.7 Configuration Management

Projects deal with many evolving items produced and used by many people sometimes involving complex dependencies. In particular during the project various versions of a software product are created. In order to avoid costly re-work, hidden defects, and deliver a software system consistent with the stage of development and traceable to the needs of the customer, configuration management is needed. Configuration management involves the development and application of procedures and standards for managing the evolution of a software product.

2.7.1 Configuration Item and Rationale

Each project churns out a number of documents, but not all are required for continued development and system maintenance. Only some baselined documents are to be kept under configuration control. This section should provide the rationale behind selection of those artifacts that would be managed for changes. Artifacts suited for configuration control include plans, specifications, designs, code, tests, manuals, object code, defect reports and change requests. Artifacts that are bad choices for configuration control include test results, project tracking results, working documents and samples.

The fewer the number the better, but all artifacts that are prone to frequent changes, have many dependencies and/or affect project progress should be considered possible configuration management items (CIs). All CIs should be classified into categories for ease of management. Categories should be based on the typical rate of change, impact of changes and/or project structure. Typical categories include project subsystems, source code, objects, tests and documents. Categorization helps in proper organization of software artifacts and enables quicker updates and retrieval.

2.7.2 Identification System

It is essential to be able to unambiguously identify a particular version of a particular artifact. A naming scheme also allows easy retrieval of CI for maintenance and reuse. The naming scheme should be general enough to cover all CIs, flexible enough to allow new CIs to be added and simple enough to be navigated and understood. An example scheme consists of the project name followed by the subsystem name followed by the specific items name and version. E.g. HVM/VT/code/generator/1.0

Another example scheme is to use numbers and allocate ranges of numbers to each CI category. For example, items can be identified by a five-digit code followed by a two-digit version number, 102-06-10, perhaps involving the date and responsible party. Use of numbers is difficult for subsequent retrieval but is flexible and lends itself easily for organization.

2.7.3 Storage of Configuration Items

The storage location for each CI should be identified. A directory structure for the CI storage should be formulated based on the directory structure used for other development artifacts (e.g. other MBASE models) and specific disks/machines should be identified for storage. A mechanism to archive previous versions should be identified and a plan for backup of project artifacts should be defined. Specific personnel assigned for the tasks of backup and archival should be identified.

577b Guidelines:

Use of Rational Corporation's ClearCase.
2.7.4 Configuration Control

Projects should identify the possible range of changes, and define a process or policy to carry out the change requests. Typically, change control requires:

- An understanding and analysis of the requested change.
- Impact and Feasibility of carrying out the change
- Authorization to carry out the change
- Implementation of the change
- Verification of the changes implemented
- Reconfiguration and baseline of software product.

The policy should address technical, management and quality issues in implementing the changes. Overall, the policy should identify steps for change analysis and change execution. Note that all the steps may not be required for every instance. It is sufficient to identify the steps for each category of possible change. Generally a configuration team conducts the analysis for a possible (complex) change, whereas the development/maintenance team carries out the change. A representation of the policy in the form of a workflow diagram is often useful.

2.7.5 Status and Accounting

Accounting activities record and report the status of project CIs. Specifically the following should be addressed:

- The CIs to be tracked and reported for changes
- The types of reports to be generated along with the frequency of generating them

At least the initial version, status of requested changes and the implementation of approved changes should be tracked. This can be reported in a simple table within a document/file, or with more sophisticated revision control tools such as CVS. Whatever is used should be clearly documented and used consistently and inspected regularly.

2.7.6 Baselining Events

Each document or software artifact is baselined at certain major project milestones. These milestones mark the overall consensus of the project’s progress and satisfaction or achievement of certain quality goals. Baselining documents puts these under configuration control when the configuration personnel take over ownership of the artifact from the developer/author. This event should be defined for each CI, as it is critical in identification of responsibility in the event of a change. The milestones should match process anchor points.

2.7.7 Resources and Personnel

Identify the software and human resources required to perform configuration management (CM) functions.

Designate specific project personnel to perform the tasks of CM and main project librarian or archivist.

2.7.8 Tools

Each of the tasks mentioned above may necessitate the use of tools for efficiency and effectiveness.
577b Guidelines:  
Use of ClearCase; "Comments" used to show baselined versions; Strict use: IV&V’ers rely on ClearCase to be assured they have the latest version.

2.8 Testing Guidelines

2.8.1 Testing Requirements

- Identify and describe the major guidelines and standards to be used in the test-related activities (planning, design, execution and evaluation of tests on the software system):
  - Test Case Norm: the types of test cases that should be developed (valid, invalid, boundary, etc).
  - Describe what kind of measures you will use to determine the progress of test activities (what type of defect counts are going to be used, how to measure successfully executed test cases).
  - Testing approach: Identify the approach to be followed to ensure adequate testing of the feature groups. Examples include:
    - Interface Testing (alpha, beta user testing)
    - Performance Testing
    - Security Testing
    - Regression Testing

577b Guidelines:  
It is acceptable to create and execute only Interface Testing plans for their projects.

- Test Case Naming Convention: how each kind of entity (such as test case and test procedure) should be named.
- Test Design Guidelines: how test procedures and, or scripts will be developed, e.g., with the underlying goals for modularity, for reuse and maintenance.
- Test Data Standards: how data will be selected or created and restored to support testing.

2.8.2 Deliverables

- Identify the expected deliverable results from the various tests (e.g. test plan(s), test description and results, problem reports using the GANT online system)
- Scripts written for testing
- Test tools if built by the team for the project.

2.8.3 Tools

- Describe tools to be used or built for testing
2.9 Defect and Change Management

Changes are a normal part of software development. It is required to have a plan in place for managing changes before they are actually required. Defect and changes often arise from several stakeholders. This section describes the practices and procedures for reporting, tracking and resolving the problems identified in peer review and testing and also in the maintenance process.

2.9.1 Reporting procedure

Identify the means by which problems and changes get reported. Problems and changes could be identified by the developers, test team or the customer. There should be a single interface for all stakeholders and by unifying the change reporting with defect reporting, the change traffic can be managed under a single umbrella.

Provide the structure and layout of the Change report form along with the process of communicating this information. It should be sufficient to meet the needs of the change control policy described in section 7.4. Document the rationale for each decision taken.

2.9.2 Tracking

The purpose of a problem and change tracking system is to ensure that these get adequately addressed within the scope of the project and at the same time serve as a means to evaluate the progress and quality of work in the project. It also provides the developer and customer feedback about the status of a problems resolution. There are many possible effective tracking systems and several commercial tracking systems. In particular the GNU foundation provides a popular free tracking system called GNATS (see http://www.gnu.org).

2.9.3 Resolution

Define the procedure whereby valid problems and changes are implemented in a time bound fashion or a justification is provided for why the changes cannot be carried out. The process should result in a Win–Win for all involved stakeholders.

3. Appendix

- The appendix should list the vendor documents, user manuals and hyperlinks for tools to be used in quality management. It should also describe the layout and structure of the forms to be used for quality management.
- Problem reporting form (directly or by reference)
- Examples of Peer Review forms (directly or by reference)

Also listed should be the coding standards defined previously in section 2.2.
3.1 Appendix with Problem Reporting Form/System

For CS577b 2002 the Problem Reporting System Form is available through a link on the Class webpage under "Tools".

3.2 Appendix with Agile Artifact Review Form Set and Instructions

For CS577b 2002 the Agile Artifact Review form and field descriptions are available through a link on the Class webpage under "Tools".

3.3 Appendix with Fagan's Inspection Form Set and Instructions

For CS577b 2002 the Fagan's Inspection form set and field descriptions are available through a links on the Class webpage under "Tools".
Test Plan

A. Description

1. Purpose

- To prescribe the scope, approach, resources, and schedule of testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with testing.

- To detail the activities required to prepare for and conduct the system test

- To communicate to all responsible parties the tasks which they are to perform, and the schedule to be followed in performing the tasks

- To define the sources of the information used to prepare the plan

- To define the test tools and environment needed to conduct the system test

2. Timing

An initial test plan is usually created during Inception or just after LCA, on a risk driven basis, and then refined during each construction iteration based on need.

3. Intended Audience

4. Participants

5. Responsibility

The test designer is responsible for planning the test effort during the project and during each iteration.

6. Completion Criteria

7. Additional Information

577b Guidelines:

8. High-Level Dependencies

- Life Cycle Plan
- Quality Management Plan
Guidelines for MBASE

Test Plan

- System and Software Requirements Definition (SSRD)
- Configuration Management Plan
- System and Software Architecture Description (SSAD)
- Relevant organizational policies
- Relevant organizational standards

<table>
<thead>
<tr>
<th>OCD</th>
<th>SSRD</th>
<th>SSAD</th>
<th>LCP</th>
<th>FRD</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

B. Document Section

1. Introduction

1.1 Purpose of the Test Plan
- Provide the purpose, background, and scope of testing within this project.

1.2 References
- Provide complete citations to all documents, meeting results, and external tools referenced or used in the preparation of this document and their outputs.
- This should be done in such a manner that the process and information used can be traced and used to reconstruct the document if necessary
- References to the following documents, when they exist, are required in the highest-level test plan (LCP 2., SSRD, SSAD)

577 Guidelines:
A "complete citation" for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]

1.3 Change Summary

577 Guidelines:
For versions of the Test Plan since the last ARB, include a summary of changes made in the document to ease the review process.

2. Environment Preparation

Specify how to prepare test environment for testing.
2.1 Hardware preparation

Describe the procedures needed to prepare the hardware for the test, including support hardware (e.g., test equipment). Reference any operating manuals, if applicable.

Provide the following as applicable:

a. The specific hardware to be used, identified by name and, if applicable, number
b. Any particular settings and cabling needed to connect the hardware
c. One or more diagrams to show hardware, interconnecting control, and data paths
d. Step-by-step instructions for placing the hardware in a state of readiness for testing

2.2 Software preparation

Describe the procedures needed to prepare the software for the test, including support software (e.g., simulators, data recording/reduction software). Reference any software manuals, if applicable.

Provide the following as applicable:

• The specific software to be used, identified by name and, if applicable, version number
• The storage medium of the software (magnetic tape, diskette)
• Instructions for loading the software, including required sequence
• Instructions for software initialization common to more than one test case

2.3 Other pre-test preparations

Describe any other pre-test personnel actions, preparations, or procedures needed to perform the test not accounted for in 4.1 or 4.2.

3. Test Identification

This section should contain the following information:

3.1 Test Identifier

This identifies the test by a project unique identifier. It shall provide a brief description of the test.

3.1.1 Test Level

This section shall describe the level at which the testing will be performed, for example, software item level or system qualification level.

3.1.2 Test Class

This section shall describe the type or class of the test that will be performed, for example, timing tests, erroneous tests, maximum capacity tests, etc.
3.1.3 Requirements Traceability

This section shall include traceability from each test identified to the software item requirements and if applicable, software system requirements it addresses.

Note:
The following template shows a way of organizing Section 3.x (this depends on whether there are more than one tests):

3.1 Test Identifier 1
   3.1.1 Test Level
   3.1.2 Test Case
   3.1.3 Requirements Traceability

3.2 Test Identifier 2
   3.2.1 Test Level
   3.2.2 Test Case
   3.2.3 Requirements Traceability

...

4. Resources

Specify the people, time, budget, resources allocated for testing, etc.

4.1 Responsibilities

Identify the groups responsible for managing, designing, preparing, executing, witnessing, inspecting and resolving test items. In addition, provide the groups responsible for providing items to be tested.

4.2 Staffing and Training Needs

Specify test-staffing needs by skill level. Identify training options for providing necessary skills.

4.3 Schedule for Testing Activities

Include test milestones. Estimate the time required to do each testing task, and a schedule for the testing activities.

4.4 Other resource allocations

Include all other resources required for the testing activities.

5. Test Completion Criteria

A statement identifying recommended test completion and evaluation criteria. Overall, individual tests may have separate completion criteria.
Test Description and Results

A. Description

1. Purpose

- Detail the activities required to prepare for and conduct specific system tests and document their results.
- Communicate to all responsible parties the tasks which they are to perform, and the schedule to be followed in performing the tasks
  - Test procedure specification. A document specifying a sequence of actions for the execution of a test.
  - Tester. Identify the person who prepared the test description.
  - Test Preparation. Identify the test tools to be used; the names of files in which test cases and/or data reside; the hardware and software required for the test.
  - Test Initialization. Describe how to set up the conditions for the test; identify any flags, breakpoints, or data to be set/reset prior to starting the test.
  - Test Procedure.

2. Timing

3. Intended Audience

4. Participants

5. Responsibility

The test designer/manager is responsible for planning the test effort during the project and during each iteration.

6. Completion Criteria

7. Additional Information

577b Guidelines:

8. High Level Dependencies

Test cases are obtained from the following sources:

- CDC Test Plan
- System Requirements (SSRD)
B. Document Sections

1. Introduction

1.1 Purpose

• Provide the purpose, background, and scope of test descriptions and results within this project.

1.2 References

• Provide complete citations to all documents, meeting results, and external tools referenced or used in the preparation of this document and their outputs.

• This should be done in such a manner that the process and information used can be traced and used to reconstruct the document if necessary

577 Guidelines:

A "complete citation" for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]

1.3 Change Summary

577 Guidelines:

For versions of the Test Description and Results since the last ARB, include a summary of changes made in the document to ease the review process.

2. Test Identification

Describe the test items including their version/revision level. A test consists of a set of one or more test cases, or a set of one or more test procedures, or a set of one or more test cases and procedures.

3. Test Preparation

This section should contain the following: Safety precautions, marked by WARNING or CAUTION, and security and privacy protection considerations shall be included as applicable

3.1 Test Identifier

This identifies the test by a project unique identifier. It shall provide a brief description and should contain the following information. References to information stated earlier can be made.

3.1.1 Hardware preparation

Describe the procedures needed to prepare the hardware for the test, including support hardware (e.g., test equipment). Reference any operating manuals, if applicable.
Provide the following as applicable:

- The specific hardware to be used, identified by name and, if applicable, number
- Any switch settings and cabling needed to connect the hardware
- One or more diagrams to show hardware, interconnecting control, and data paths
- Step-by-step instructions for placing the hardware in a state of readiness

### 3.1.2 Software preparation

Describe the procedures needed to prepare the software for the test, including support software (e.g., simulators, gages, data recording/reduction software). Reference any software manuals, if applicable.

Provide the following as applicable:

- The specific software to be used, identified by name and if applicable version number
- The storage medium of the item(s) under test (magnetic tape, diskette)
- The storage medium of any related software (simulators, test drivers, databases)
- Instructions for loading the software, including required sequence
- Instructions for software initialization common to more than one test case

### 3.1.3 Other pre-test preparations

Describe any other pre-test personnel actions, preparations, or procedures needed to perform the test.

**Note:**

The following template shows a way of organizing Section 3.x (this depends on whether there are more than one tests):

```
3.1 Test Identifier 1
   3.1.1 Hardware Preparation
   3.1.2 Software Preparation
   3.1.3 Other pre-test preparations

3.2 Test Identifier 2
   3.2.1 Hardware Preparation
   3.2.2 Software Preparation
   3.2.3 Other pre-test preparations

...
```

### 4. Test Case Description

A test case specification specifies inputs, predicted results, and a set of execution conditions for a test item. For each test case, create a sub-heading using the following structure:

Identify a test (one of the tests in the test set comprising the application testing addressed by this test description) by project-unique identifier and provide the information specified below for the test. The name includes the identification of the applicable unit.
4.1 Test Identifier

This identifies the test by a project unique identifier. It shall provide a brief description and should contain the following information. References to information stated earlier can be made.

**Note:**

This section should map to the corresponding section 3.1 Test Identifier.

4.1.1 Test Case x. – (Project-unique identifier)

This identifies the test case by a project unique identifier. It shall provide a brief description and should contain the following information. References to information stated earlier can be made.

4.1.1.1 Test Items

Identify and briefly describe the test items and features to be exercised by this test case. That may also include nonfunctional or extra-functional requirements, such as performance.

For each item, consider supplying references to the following documentation:

1. Requirements specification
2. Design specification
3. Users guide
4. Operations guide
5. Installation guide

4.1.1.2 Pre–conditions

Describe prerequisite conditions that must be established prior to performing the test case, such as flags, initial breakpoints, pointers, control parameters, or initial data to be set/reset prior to test commencement.

4.1.1.3 Post–conditions

Describe conditions that must be established after performing the test case such as resetting data, logging out, removing monitors, breakpoints, etc.

4.1.1.4 Input Specifications

Specify each input required to execute the test case. Some of the inputs may be specified by value (with tolerances or choices where appropriate), while others such as constant tables or transaction files, will be specified by name.

The following shall be provided as applicable:

1) Name, purpose, and description (e.g., range of values, accuracy) of each test input
2) Source of the test input and the method to be used for selecting the test input
3) Whether the test input is real or simulated
4) Time or event sequence of test input

5) The manner in which the input data will be controlled to:
   a) Test the item(s) with a minimum/reasonable number of data types and values
   b) Exercise the item(s) with a range of valid data types and values that test for overload, saturation, and other “worst case” effects
   c) Exercise the item(s) with invalid data types and values to test for appropriate handling of irregular inputs
   d) Permit retesting, if necessary

4.1.1.5 Expected Output Specifications

Identify all expected test results for the test case, both intermediate and final test results, as applicable.

4.1.1.6 Pass/Fail Criteria

Identify the pass/fail criteria to be used for evaluating the intermediate and final results of this test case. The following shall be provided as applicable:

1) The range or accuracy over which an output can vary and still be acceptable
2) Minimum number of combinations or alternatives of input and output conditions that constitute an acceptable test result
3) Maximum/minimum allowable test duration, in terms of time or number of events
4) Maximum number of interrupts, halts, or other system breaks that may occur
5) Allowable severity of processing errors
6) Conditions under which the result is inconclusive and retesting is to be performed
7) Conditions under which the outputs are to be interpreted as indicating irregularities in input test data, in the test database/data files, or in test procedures
8) Allowable indications of the control, status, and results of the test and the readiness for the next test case (may be output of auxiliary test software)
9) Additional criteria not mentioned above

4.1.1.7 Test process

Define the test process for this test case. The test process is a series of individually numbered steps listed sequentially in the order in which the steps are to be performed. If the process is non-trivial, include a Procedure Template. You may reference a separate Test Procedure Specification (section 4.x) if the Procedure Specification applies to multiple Test Cases.

Test Procedure Template:

A test procedure provides detailed steps that carry out the test as defined by the associated test case(s). A sample test procedure template is shown in Figure 10.

Figure 10: Test Procedure Template
Provide the following for each test procedure, as applicable:

1) Test operator actions and equipment operation required for each step, including commands, as applicable to:
   a) Initiate the test case and apply test inputs
   b) Inspect test conditions
   c) Perform interim evaluations of test results
   d) Record data
   e) Halt or interrupt the test case
   f) Request data dumps or other aids, if needed
   g) Modify the database/data files
   h) Repeat the test case, if unsuccessful
   i) Apply alternate modes as required by the test case
   j) Terminate the test case

2) Test Inputs

3) Expected result and evaluation criteria for each step

4) If the test case addresses multiple requirements, identification of which test procedure step(s) address which requirements.

5) Actions to follow in the event of a system stop or indicated error, such as:
   a) Recording of critical data from indicators for reference purposes
   b) Halting or pausing time-sensitive test-support software and test apparatus
   c) Collection of system and operator records of test results

6) Procedures to be used to validate expected results, reduce and analyze test results to accomplish the following, as applicable:
   a) Detect whether an output has been produced
   b) Identify media and location of data produced by the test case
   c) Evaluate output as a basis for continuation of test sequence
   d) Evaluate test output against required output

### 4.1.1.8 Assumptions and constraints

Identify any assumptions made and constraints or limitations imposed in the description of this test case due to system or test conditions, such as limitations on timing, interfaces, equipment, personnel, database/data files.
4.1.1.9 Dependencies

List the identifiers of the test cases, which must be executed prior to this test case or for that depend on the execution of this test. Summarize the nature of the dependencies.

4.1.1.10 Traceability

Include reference to:

- SSAD, OCD addressed
- Use Case/Scenario addressed

Note:
The following template shows a way of organizing Section 4.x (this depends on whether there are more than one tests and each test has several test cases):

4.1 Test Identifier 1
  4.1.1 Test Case 1
    4.1.1.1 Test Item
    4.1.1.2 Pre-Conditions
    4.1.1.3 Post-Conditions
    4.1.1.4 Input Specifications
    4.1.1.5 Expected Output Specifications
    4.1.1.6 Pass/Fail Criteria
    4.1.1.7 Test Process
    4.1.1.8 Assumptions and Constraints
    4.1.1.9 Dependencies
    4.1.1.10 Traceability

4.1.2 Test Case 2
  4.1.2.1 Test Item
  4.1.2.2 Pre-Conditions
  4.1.2.3 Post-Conditions
  4.1.2.4 Input Specifications
  4.1.2.5 Expected Output Specifications
  4.1.2.6 Pass/Fail Criteria
  4.1.2.7 Test Process
  4.1.2.8 Assumptions and Constraints
  4.1.2.9 Dependencies
  4.1.2.10 Traceability

4.2 Test Identifier 2
  4.2.1 Test Case 1
    4.2.1.1 Test Item
    4.2.1.2 Pre-Conditions
    4.2.1.3 Post-Conditions
    4.2.1.4 Input Specifications
    4.2.1.5 Expected Output Specifications
    4.2.1.6 Pass/Fail Criteria
    4.2.1.7 Test Process
    4.2.1.8 Assumptions and Constraints
    4.2.1.9 Dependencies
    4.2.1.10 Traceability

...
5. Test Incident Reports

A test incident report is a document reporting on any event that occurs during the testing process, which requires further investigation.

5.1 Identifier

Specify the unique identifier assigned to this test incident report.

5.2 Summary

Briefly summarize the incident. Identify the test items involved indicating their version/revision level. References to the appropriate test-procedure specification, test-case specification, and test log should be supplied.

5.3 Incident Description

Provide a detailed description of the incident. This description should include the following items:

• Inputs
• Expected results
• Actual results
• Anomalies
• Date and time
• Procedure step Environment
• Attempts to repeat
• Testers
• Observers

Related activities and observations that may help to isolate and correct the cause of the incident should be included.

For example, describe any test-case executions that might have a bearing on this particular incident and any variations from the published test procedure.

5.4 Impact

If known, indicate what impact this incident will have on test plans, test-design specifications, test-procedure specifications, or test-case specifications.

6. Test Log

The purpose of the test logs is to provide a chronological record of relevant details about the execution of tests.

6.1 Test-Log Identifier

Specify the unique identifier and description name assigned to this test log.
6.2 Test Record

This section shall present, possibly in a figure or annex, a chronological record of the test events covered by this report. This test log shall include:

a) The date(s), time(s), and location(s) of the tests performed

b) The hardware and software configurations used for each test including, as applicable, part/model/serial number, manufacturer, revision level, and calibration date of all hardware, and version number and name for the software components used

c) The date and time of each test-related activity, the identity of the individual(s) who performed the activity, and the identities of witnesses, as applicable

7. Test Summary

This provides an overview of test results.

7.1 Overall assessment of the software tested:

This section shall:

a) Provide an overall assessment of the software as demonstrated by the test results in this report.

b) Identify any remaining deficiencies, limitations, or constraints that were detected by the testing performed. Problem/change reports may be used to provide deficiency information.

c) For each remaining deficiency, limitation, or constraint, describe:
   1) Its impact on software and system performance, including identification of requirements not met.
   2) The impact on software and system design to correct it.
   3) A recommended solution/approach for correcting it.

7.2 Impact of test environment:

This section shall provide an assessment of the manner in which the test environment may be different from the operational environment and the effect of this difference on the test results.

7.3 Recommended improvements:

This section shall provide any recommended improvements in the design, operation, or testing of the software tested. A discussion of each recommendation and its impact on the software may be provided. If no recommended improvements are provided, this shall state “None”.

8. Notes

Provide any general information and rationale pertinent to the information contained in this test description and results report. Include a list of acronyms and abbreviations, and a list of terms and their definitions used in this test report. This list contains definitions needed to understand this test report. Include additional definitions and delete those not applicable.
9. Appendices

Use appendices to provide information that is published separately or that does not fit conveniently in the body of the document. Each appendix should be referenced in one of the above sections where data would normally have been provided.

Appendix 1: Test log

Present a chronological record of software requirement verification activities, indicating the following:

- Date
- Time
- Location
- Participants
- Reference (Test Case/Test Report/Peer Review Report, etc…)

Peer Review Plan

A. Description

1. Purpose

The peer review plan describes the peer [or people-based] review process(es) and actual peer review checklists or Fagan's Inspection exit criteria (or references to them), and identifies the people involved and the schedule of peer reviews for the project. Since IV&V reviews are also "people-based" reviews, they should be covered by this document, although it may not be possible to provide all the same information.

2. Timing

A draft of this plan is expected to be completed prior to RLCA, and updated as needed during Construction. Prior to RLCA, the salient points are expected to be documented in the LCP's Quality section.

3. Intended Audience

This plan is for the team-members (for personal planning), and the organization's Quality Oversight organization.

4. Participants

The quality person/role usually participates in generating and updating this document. The project manager usually assist in the "risk driven" decision making.

5. Responsibility

The project manager should ensure that a Peer Review Plan is created, he/she may delegate it to the Quality Focal Point to create the peer review plan, designate areas for peer reviews on a risk driven basis and facilitate persons performing the peer reviews.

6. Completion Criteria

7. Additional Information

577b Guidelines:

The class website should be consulted for the latest versions of the instruments on which to record quality metrics data.

8. High-Level Dependencies

<table>
<thead>
<tr>
<th>OCD</th>
<th>SSRD</th>
<th>SSAD</th>
<th>LCP</th>
<th>FD</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>+</td>
<td>+</td>
<td>√</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>
B. Document Sections

1. Purpose

1.1 Overview

Describe the intended purpose of the in-process, people-based reviews and expected outcomes for this project.

577b Guidelines:

1.2 Since there can be both team and independent (IV&Ver) in-process, people-based reviews for CS577, both should be described if you have an IV&Ver assigned to your project. For sections 2 through 4, this can be done using two separate sub-sections.

1.3 References

This document should refer to external peer review documents used by the project (e.g. list any Inspection process used). References made to all documents along with their versions and numbers should be identified in this section.

577 Guidelines:

A “complete citation” for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]

1.4 Change Summary

For each version of the Peer Review plan, describe the main changes since the previous version. The goal is to help a reviewer focus on the most critical parts needing review.

577 Guidelines:

For versions of the Peer Review plan since the last ARB, include a summary of changes made in the document to ease the review process.

2. Peer Review Items, Participants and Roles

- Describe the artifacts (documents, etc…) that will be subjected to peer review, taking into account schedule or staffing constraints

- For each artifact, indicate the primary author, and identify who will play the various roles. This information will be required to help the participants involved prepare for their respective roles.

- Provide checklists for reviews or exit criteria for inspections for each of the artifacts to be used during the peer review process. An example is the JPL Inspection Process reference checklists.

- You may simply provide a reference to the various Peer Review Announcements (to be included in the Appendix)
3. Peer Review Milestones

Provide details about the schedules for the various activities for peer reviews:

- completion of planning
- overview meeting
- preparation
- peer review meeting (date, time, place)
- peer review data summary and results reporting

4. Peer Review Process

- Describe the peer review process for each type of item to be reviewed
- Reference any inspection process (e.g., Fagan Code Inspection): if that is the case, simply indicate any variations introduced particularly for your project, such as Perspective Based Reading or Object Oriented Reading Technique.

5. Classification of Defects

All defects identified during the peer review meeting shall be classified by severity and type. This serves as a metric for future project. Example severity levels and type of defects are presented in the tables below. Consider using existing defect classifications such as Orthogonal Defect Classification (ODC). Describe the classification scheme to be used for this project. In addition to an overall description, define the form below from the project perspective.

5.1 Severity

Major

Define what the major defects are. You may wish to specialize the following:

- A condition that causes an operational failure, malfunction, or prevents attainment of an expected or specified result
- Information that would lead to an incorrect response or misinterpretation of the information by the user
- An instance of non-conformance that would lead to a discrepancy report if implemented as is

Minor

- A violation of standards, guidelines, or rules, but would not lead to a discrepancy report
- Information that is undesirable but would not cause a malfunction or unexpected results (bad workmanship)
- Information that, if left uncorrected, may decrease maintainability

5.2 Type

Define the type of defect
5.2.1 Missing

Information that is specified in the requirements or standard, but is not present in the document

5.2.2 Wrong

Information that is specified in the requirements or standards and is present in the document, but the information is incorrect

5.2.3 Extra

Information that is not specified in the requirements or standards but is present in the document

5.3 Category

There are two types defects that need to be addressed: those that are identified objectively and those that are identified as defects because they resulted in changes.

5.3.1 Category for Defects Identified Objectively

Those defects that are identified against objective criteria by an in-process review with the author present should be categorized to provide information for subsequent causal analysis and/or process improvement suggestions.

- Refine the types of defects to the peer review items identified in Section 5.2
- Examples of defect types include:
  - Logic
  - Syntax
  - Clarity
  - Performance
  - Interface
  - No Error Handling
  - ...

5.3.2 Category for Defects Which Are Classified as Such After the Review

Any defect identified as such AFTER the actual review (either because they are issues in another document or were identified without the author present (and therefore his/her assumed concurrence), should be further categorized as

- Unavoidable. Unavoidable defects (AKA changes) arise because of the methods, techniques or approaches being followed necessitate changes. Such situations are as a result of life cycle and project dynamics. Examples include changes arising because of
  - the dynamics of learning,
• exploration in IKIWISI situations,
• code or screen contents reorganizations taken on as an "afterthought",
• replacement of stubs or placeholders in code, etc.

• Avoidable. Avoidable defects are those changes in analysis, design, code or documentation arising from human error, and which could be avoided through better analysis, design, training, etc. Examples include
  • stub replacement that violates win conditions or requirements such as execution time or memory space. For instance, the replacement of a "stub" which breaks a critical timing constraint.

6. Schedule for Peer Reviews

This section describes the planned schedule for the artifacts that will be subject to Peer Reviews of any type, whether done by IV&V or the team. It should be tied to the schedule in the LCP. If some or all of the information requested below is covered by the LCP plan, provide an explanation of where it is and how to identify it.

The planned schedule can be documented with a simple table that identifies what quality assessment techniques are to be applied when to what documents.

<table>
<thead>
<tr>
<th>What Artifact</th>
<th>When</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Y</td>
<td>1 week before CCD</td>
<td>Walkthrough [Informal Review]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Test Plan</td>
<td>2 weeks after RLCA</td>
<td>Fagan's Inspection</td>
</tr>
</tbody>
</table>

7. Appendices: Forms for Peer Review Data Gathering

Include either the forms or references to them for ALL the forms and instruction sets to be used in gathering information from the Peer Review activities on your project.

Appendix 2: Fagan's Inspection Data Reporting Forms

Include in the appendix references to the forms to be used by the various participants in the preparation and execution of Fagan's Inspections. These include:

• Announcement
• Individual Preparation Log
• Defect List
• Detailed Report
• Summary Report
The set forms and field descriptions for a Fagan's inspection are included in the spreadsheets available at ....

7.1 Agile Data Reporting Forms

This set of forms is on the following worksheets of "AgileArtifactReview_Form_v6.xls"

- Areas of Concern Log
- Problem List Cover
- Problem List
**Figure 10: Area of Concern Log Form**

<table>
<thead>
<tr>
<th>Project Name:</th>
<th>Review Id:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact:</td>
<td>Review Date:</td>
</tr>
<tr>
<td>Module:</td>
<td>Review Time:</td>
</tr>
<tr>
<td>MBASE Phase/level:</td>
<td>Exit Criteria:</td>
</tr>
<tr>
<td>Activity:</td>
<td>Date Sent to Reviewer:</td>
</tr>
<tr>
<td>Review Leader:</td>
<td>Date Prep. Completed:</td>
</tr>
<tr>
<td>Review Leader email:</td>
<td>Date Returned to Author:</td>
</tr>
<tr>
<td>Review Leader phone:</td>
<td>Date Returned to QAT:</td>
</tr>
<tr>
<td>Author:</td>
<td>Reviewer 1:</td>
</tr>
<tr>
<td>Reviewer 2:</td>
<td>Reviewer 3:</td>
</tr>
<tr>
<td>Reviewer 4:</td>
<td>Reviewer 5:</td>
</tr>
<tr>
<td>Reviewer 6:</td>
<td>Reviewer 7:</td>
</tr>
</tbody>
</table>

Use this sheet to record the areas of concern that come up during your reading/review of the Artifact. Give the "location" information and the associated technical description of the area of concern to indicate to the developer/author during his/her analysis of this information about the relevant part of the Artifact. Give your opinion for the classification of the area of concern in Mm/MWE field. Write in letter each for Major/Minor (Mm) and for Missing/Wrong/Extra (M/W/E).

Keep this sheet with you during the analysis of the artifact. When an area of concern you recorded here requires corrective action and is placed on the artifact's Problem list, note the number of the Defect/Issue(s) in the Areas of Concern Log. Problems are things you believe the author of this artifact can/should fix; "open issues" are things which can not be corrected solely in this artifact or at this time.

<table>
<thead>
<tr>
<th>#</th>
<th>Location(s)</th>
<th>Area of concern</th>
<th>Mm</th>
<th>M/W/E</th>
</tr>
</thead>
</table>

© 2002 A W Brown BES/MBSE USC CSE  
AgileInternalReview_form(exC/MBASE/TeSe-v1.2a) -- Areas of Concern Log  
1/9/2003
### Figure 11: Problems/Issuer Log Cover Sheet Form

<table>
<thead>
<tr>
<th>Problem Description</th>
<th>Classification</th>
<th>Cost</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- Major
- Minor
- Open Issue
- Extra
- Missing
- Wrong
- Avoidable
- Expected
- New
- Unknown
- Mini
- Default

**Column Headings:**
- Description
- Classification
- Cost
- Notes
The field descriptions for the forms, which are shown in the Table 36 and Table 37, can also be found in the following worksheets of Microsoft Excel spreadsheet in the file named "AgileArtifactReview_Form-FieldDesc_v6.xls".

- Areas of Concern Log Field Desc
- Problem List Field Description
Table 36: Field Descriptions for the Areas of Concern Log

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Name</strong></td>
<td>Insert the project name or title. For purpose of the class, it should be the team name, such as Team 3.</td>
</tr>
<tr>
<td><strong>Artifact</strong></td>
<td>Identify the name of the artifact and/or its section numbers being reviewed e.g. OCD or SSRD 2.1 to SSRD 2.3</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>Identify the name of the software/code module and/or its range of the line numbers.</td>
</tr>
<tr>
<td><strong>MBASE Phase/level</strong></td>
<td>The phase in which the component or artifact was generated. E.g. Inception, Elaboration, Construction or Transition. If there are cycles within a phase, then please indicate which cycle also.</td>
</tr>
<tr>
<td><strong>Activity</strong></td>
<td>Identify the activity in which this review is being performed e.g. requirement, design, implementation etc. This also corresponds to the activity referred to in the effort report.</td>
</tr>
<tr>
<td><strong>Exit Criteria</strong></td>
<td>Identify the name of the document that specifies exit criteria of this review including full path and file name, URL, or attach a printed copy.</td>
</tr>
<tr>
<td><strong>Review Leader</strong></td>
<td>Name of person heading the &quot;review&quot; i.e. the person responsible for the review meeting and turning in the forms.</td>
</tr>
<tr>
<td><strong>Review Leader email</strong></td>
<td>Email address of person heading the &quot;review&quot;.</td>
</tr>
<tr>
<td><strong>Reviewer phone</strong></td>
<td>Phone number of person heading the &quot;review&quot;.</td>
</tr>
<tr>
<td><strong>Author</strong></td>
<td>Name of last updating the artifact; this person is also the one responsible for fixing the problems found.</td>
</tr>
<tr>
<td><strong>Review if</strong></td>
<td>Identify the review number. It is composed of two parts, one to identify the &quot;Artifact&quot; or &quot;Module&quot; and the other to identify the number of reviews performed on the module. For example if it is second review of the OCD, OCD - 2. The running number starts at 1 at each phase.</td>
</tr>
<tr>
<td><strong>Review Date</strong></td>
<td>Identify the date(s) that the review meeting was conducted.</td>
</tr>
<tr>
<td><strong>Review Time</strong></td>
<td>Identify the start and end time(s) that the review meeting was conducted.</td>
</tr>
<tr>
<td><strong>Dates:</strong></td>
<td>NOTE: The following information is desired for tracking purposes only.</td>
</tr>
<tr>
<td><strong>Date Sent To Reviewer</strong></td>
<td>Date when sent to reviewer(s), or when reviewers told to retrieve the item.</td>
</tr>
<tr>
<td><strong>Date Prep. Completed</strong></td>
<td>Date when the reviewer(s) completed their individual preparation.</td>
</tr>
<tr>
<td><strong>Date Returned to Auth</strong></td>
<td>Date when reviewers returned the Log to the &quot;Author&quot;; usually at the meeting.</td>
</tr>
<tr>
<td><strong>Date Returned to QAT</strong></td>
<td>Date when the &quot;Author&quot; has finished recording defect/issue determination and defect repair information and returned to the Quality Assessment Tracker</td>
</tr>
<tr>
<td><strong>Reviewer(s)</strong></td>
<td>Place the name or initials of each participating reviewer and his/her preparation time in parenthesis e.g. A/NB (xx mins)</td>
</tr>
<tr>
<td><strong>Column Heading</strong></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>A running number of area of concern used just to ease the counting and avoid losing information.</td>
</tr>
<tr>
<td>Location(s)</td>
<td>If document, identify section, paragraph, and/or line/sentence number of area of concern. If program, identify file name, subprogram, and physical/logical line number.</td>
</tr>
<tr>
<td>Area of concern</td>
<td>A description of the concern, which may become either a defect or an issue.</td>
</tr>
<tr>
<td>Mm</td>
<td>The classification of the fix (defect) as Major or Minor, or Issue as Open. Major defects are those that would result in a failure during operation or otherwise violate a client's requirements.</td>
</tr>
</tbody>
</table>
Table 37: Field Descriptions for Artifact Review Problem

### List

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Name</td>
<td>Insert the project name or title. For purpose of the class, it should be the team name, such as Team 3.</td>
</tr>
<tr>
<td>Review #</td>
<td>Identify the review number. It is composed of two parts, one to identify the module and the other to identify number of review performed on the module (including version). For example if it is second review of OCD, OCD - 2.</td>
</tr>
<tr>
<td>Artifact</td>
<td>Identify the name of the artifact with version number and/or its section numbers being reviewed e.g. OCD or SSRD 2.1 to SSRD 2.3</td>
</tr>
<tr>
<td>Date</td>
<td>Identify the date(s) the list is filled/modified.</td>
</tr>
<tr>
<td>Module</td>
<td>Identify the name of the software/code module (e.g. class etc) and/or the range of the line numbers which are being reviewed.</td>
</tr>
<tr>
<td>Type of review</td>
<td>Identify the type of review like Agile Internal, Agile Artifact etc.</td>
</tr>
<tr>
<td>MBASEPhase/level</td>
<td>Inception, Elaboration, Construction or Transition. If there are cycles within a phase, then please indicate which cycle also.</td>
</tr>
<tr>
<td>Review Date(s):</td>
<td>Identify the date(s) that the review was conducted.</td>
</tr>
<tr>
<td>Defects Found</td>
<td>Identify number of Major and Minor defects found. Major defects are those that would result in a failure during operation or otherwise violate a client's requirements.</td>
</tr>
<tr>
<td>No. of Open issues</td>
<td>Issues are problems that the &quot;Author&quot; can not fix solely in this artifact or at this time.</td>
</tr>
<tr>
<td>No. of Unavoidable defect</td>
<td>Unavoidable defects (AKA changes) arise because of the methods, techniques or approaches being followed necessitate changes. Examples include changes arising because of the dynamics of learning, exploration in IKWISI situations, code or screen contents reorganizations taken on as an &quot;afterthought&quot;, replacement of stubs or place-holders in code, etc. Such situations are often &quot;planned for&quot; and expected to occur.</td>
</tr>
<tr>
<td>No. of Avoidable defects</td>
<td>Changes in analysis, design, code or documentation arising from human error, and which could be avoided through better analysis, design, training, etc. Examples include stub replacement that violates win conditions or requirements such as execution time, memory space; for instance the replacement of a &quot;stub&quot; which breaks a critical timing constraint.</td>
</tr>
<tr>
<td>Comments:</td>
<td>Any Quality assessment issues, or process or improvement suggestions.</td>
</tr>
<tr>
<td>Defects/issue</td>
<td>An indication of the AUTHOR'S classification of the fixes; &quot;D&quot; for Defects and &quot;I&quot; for Issues. Defects are things the AUTHOR has or will fix in the document under review; Issues are things that can not be fixed in this artifact or are related to other artifacts which the author can NOT fix directly or can not be fixed at this time. [Used just to ease the counting and avoid losing information.]</td>
</tr>
<tr>
<td>#:</td>
<td>A running number beginning with &quot;D&quot; for Defects and &quot;I&quot; for Issues, used just to ease the counting and avoid losing information.</td>
</tr>
<tr>
<td>Location(s)</td>
<td>If document, identify section, paragraph, and/or line/sentence number of the defect or issue. If program, identify file name, subroutine, and line number.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the defect. Include the PBP &quot;wrong&quot; defect sub-classifications of &quot;Incorrect Fact&quot;, &quot;Inconsistency&quot;, or &quot;Ambiguity&quot;, if appropriate.</td>
</tr>
<tr>
<td>Concern #:</td>
<td>Identify the number of concern from the Area of Concern Log to which this Defect/Issue belong. This is to map the Defect/Issue reported here (at the Problem List) to the concern that was raised in the &quot;Area of Concern Log&quot;.</td>
</tr>
<tr>
<td>Classification</td>
<td>The classification of the fix (detailed) as Major or Minor, or Issue as Open. Major defects are those that would result in a failure during operation or otherwise violate a client's requirements.</td>
</tr>
</tbody>
</table>
Peer Review Report

A. Description

1. Purpose
2. Timing
3. Intended Audience
4. Participants
5. Responsibility
6. Completion Criteria
7. Additional Information

577b Guidelines:

8. High –Level Dependencies

B. Document Sections

1. Purpose

1.1 Overview

This document provides information on what peer review information is reported for the various types of peer reviews identified in the Peer Review Plan

1.2 References

This document should refer to external peer review documents used by the project. References made to all documents along with their versions and numbers should be identified in this section.

577 Guidelines:

A "complete citation" for CS577 should include the title of the document (in suitable bibliographic form), and with the explicit URL for the document. [This information is requested so that future researchers can find the cited document from an on-line archive.]
1.3 Change Summary

For each version of the Peer Review report, describe the main changes since the previous version. The goal is to help a reviewer focus on the most critical parts needing review.

577 Guidelines:

For versions of the Peer Review report since the last ARB, include a summary of changes made in the document to ease the review process.

2. Individual Inspection-like Peer Reviews

The following information is generally provided as part of any Peer Review.

2.1 Participants and Roles

- Indicate the primary author, and identify who will play the various roles for the peer review.
- Inspectors should include in the appendix the Summary Forms, Detailed Report and the Summary Report as specified in the Peer Review Plan Appendix, section 5.

577b Guidelines:

The Completion Criteria from LCP Section 2.0 can be used as guidelines for the exit criteria for each Peer Review Item.

2.2 Peer Review Items

- Describe the item(s) that were subjected to this peer review.
- Describe the exit criteria used for each Peer Review Item (and whether they were updated from Peer Review Plan, section 4.)

577b Guidelines:

The Completion Criteria from LCP Section 2.0 can be used as guidelines for the exit criteria for each Peer Review Item.

2.3 Pre-Peer Review Meeting Defect Data

Report the pre-peer review meeting data such as: time, majors and minors found per inspector.

577b Guidelines:

Indicate your tailoring.

2.4 Peer Review Meeting Defect Data

Report the meeting defect data: size of artifact, major and minor counts, time of peer review, new defects found during the meeting not found in preparation.

577b Guidelines:

Links to the Quality Reports are sufficient.

2.5 Post-Peer Review Meeting Defect Data

Report the post-meeting data: rework hours and final count of defects.
577b Guidelines:

Provide a summarized report of open defects, issues and defect count using Weekly Progress Reports. Also indicate the number of open issues and defects between anchor points.

2.6 Peer Review Statistics

From the above, compute the following peer review statistics:

- total effort consumed for each peer reviewer
- defect density of the inspected artifacts
- defects asserted per minute of the meeting
- defect removal effectiveness

These are defined as follows:

- Time meeting effort = Time Meeting * Persons
- Total defects found = Items from preparation + New items
- Defects asserted per minute of meeting = Total Defects / Time Meeting
- Peer review effort = Preparation effort + Meeting effort + rework effort
- Defect removal effectiveness = Defects found / Peer review effort
- Defect density = Defects / SLOC or Pages
- Peer review rate = Inspected pages / Meeting time

2.7 Defect Correction and Rework

Provide the status of defect correction, including

- technical approach of outstanding rework
- due date of outstanding rework if any

Provide information for tracking that the defects were properly removed.

577b Guidelines:

Since most of this information is captured on the supplied forms, it is sufficient to reference them.

3. Other types of Peer Reviews

577b Guidelines:

Provide the applicable information (similar to that above for Inspections) for any other kinds of Peer Reviews performed by your project. These other Peer Reviews should include any Agile or normal types that might be performed.
4. Peer Review Summary Information

4.1 Peer Reviews' Performed

For Artifacts prior to Construction, provide a reference to the information in the LCP §4.4.

For Artifacts during the Construction phase, including critical technical documents (e.g. SSRD and SSAD), code and possibly tests, provide information similar to that in the "Example of Table of Assessment Activities History" of LCP §4.4.

4.2 Peer Review Results

Provide any information that is deemed useful to your project to assist with closed-loop control of the project.

577b Guidelines:

At the very least, this might include

- the COQUALMO data recording form, TableSPD-8b.doc, available from the CS577b website.
- the aggregated weekly open problem report and issues list identified in the Weekly Status Report.

5. Appendix

Include in the Appendix all variations of the following forms (either directly or by reference), which are to be completed by the various participants in the in-process team-based Peer Review(s) Meeting(s) for your project.

- Announcement
- Individual Preparation Log
- Defect List
- Detailed Report
- Summary Report

Examples should range from full-up Fagan's Inspections to Agile Artifact Review Forms.

Also include in the Appendix all variations of the forms (either directly or by reference) which are to be completed by an IV&V person assigned to your project. Examples range from Formal Review to Agile Artifact Review.
Transition Plan

A. Description

1. Purpose

The purpose of the transition plan is to ensure that the system’s operational stakeholders (users, administrators, operators, maintainers) are able to successfully operate and maintain the system.

2. Timing

The plans are typically created during construction, to be used in the transition phase. A Transition Preparation Plan, providing top-level decisions for Sections 1 and 2, and construction-phase transition preparation activities, responsibilities, plans, and resources in Section 3, 4 and 5.

3. Intended Audience

The primary audiences are the transition participants (developers, operators, maintainers, operations managers, suppliers, user representatives, others as appropriate) and others involved in the Transition Readiness Review (customers, transition experts, senior management).

4. Participants

See Intended Audience above.

5. Responsibility

The Project Manager is responsible for creating and updating the transition plans, and for ensuring the commitment of the other stakeholders to them. These responsibilities may be delegated to other team members. You may consider forming a transition team.

6. Completion Criteria

The primary completion criteria are stakeholder concurrence, feasibility of execution, and assurance of successful transition. These are evaluated by the equivalent of an Architecture Review Board in a Transition Readiness Review. The plan should address the following transition success factors: hardware readiness; software readiness; site readiness; operational staff readiness; and operational test and evaluation success.

7. Additional Information

8. High-Level Dependencies

The Transition Plan draws on and elaborates the transition aspects of the Operational Concept (OCD 4); Milestones and Products (LCP 2.) and Responsibilities (LCP 3.) from the Life Cycle Plan. It prepares for the successful execution of the life cycle support activities and responsibilities in the Support Plan.
B. Document Sections

1. Transition Strategy

This section shall be divided into paragraphs as needed to describe the developer's plans for transitioning the deliverable software to the support agency. This section shall address the following:

1.1 Transition Objectives

Establish the various dimensions of success associated with the particular transition activity. These may include the following classes of transition choices:

- Extent of capability transitioned: limited pilot operation; full operation; intermediate.
  - Number and nature of transition sites: one vs. many; homogeneous vs. heterogeneous.
  - Degree of post-transition developer support: none (pure turnkey), full (facilities management by developer), intermediate.
  - Degree of validation of operational satisfaction of stakeholder objectives.
  - Nature of product transition: shrink-wrapped product; new system; improvement in existing operation.

- Relation to support objectives, e.g., data collection to support evolution planning.

1.2 Transition Process Strategy

This section establishes the major strategic transition choices:

- Phasing of cutover: instantaneous (cold turkey); incremental; parallel operation; other.
- Phasing of transition of multiple increments to multiple sites.
- Role of alpha-testing, beta-testing, independent operational testing and evaluation.

2. Preparing for Transition

Preparing for transition is a significant non-trivial task critical to ultimate successful operation of the system. Careful attention should be paid to scheduling adequate time and resources. Anticipate problems and plan for them. Be careful of risky assumptions (e.g. the customer will hire an experienced database administrator to install Oracle and set up the proper database schema)

2.1 Hardware Preparation

- Indicate additional hardware that needs to be purchased if any
- Indicate any special or additional instructions for placing the hardware in a state of readiness
• Staff time required for hardware preparation
• Organizational approvals for nonstandard hardware installation

### 2.2 Software Preparation

This section should draw upon and extend LCP 2.3, Project Deliverables, to establish:

• Appropriate packaging of deliverable support software (tools; infrastructure software).
• Licenses for commercial software.
• Preparation and installation of necessary data.
• Conversion of legacy software for compatibility with the new system.
• Organizational approvals for nonstandard software installation

### 2.3 Site Preparation

Facilities may include:

• computer rooms, flooring, power supply
• computers, peripherals, and supplies
• data communications capabilities
• security and backup facilities

### 2.4 Staff Preparation

This section shall be divided into paragraphs as appropriate to describe the developer's plans for training support personnel to support the deliverable software.

#### 2.4.1 Training Deliverables

• Hardware and Software Preparation
• Staff availability
• Impact on staff time

#### 2.4.2 Training Schedule

• Number of training sessions
• Lengths of sessions
• Contents of each session
• Training materials used in each session
2.4.3  Measure of Success

- Evaluations of the results of the training

2.5  Operational Test and Evaluation

Assurance of adequate transition readiness is essential for successful transition. Operational test and evaluation help identify risks, problem areas, set baselines, and manage stakeholder expectations. Address the following items to estimate:

2.5.1  Evaluation Criteria

Carefully define appropriate metrics to be collected and analyzed addressing such criteria as improved efficiency, quality of service, mission effectiveness, stakeholder satisfaction, and return on investment (FRD B.2.1.5)

2.5.2  Procedures

This section establishes the major operational test and evaluation procedure choices. Refer to the test plan or test description results if needed.

- Exercise procedures: operational scenarios, alpha test, beta test, other
- Exercise participants: number, qualifications, organizational representation.
- Instrumentation: performance measurement, problem reports, questionnaires, other.
- Analysis procedures with respect to evaluation criteria.

2.5.3  Outline of Operational Test and Evaluation Report

The report should include an update of the evaluation criteria and procedures above, and various summaries of the results, conclusions, and recommendations.

3.  Stakeholder Roles and Responsibilities

This should be a transition-oriented elaboration of your version of Table 2, (LCP 3.1), and its refinements in LCP 3.2.2. Transition roles and responsibilities can vary significantly, depending on the nature of the product (shrink-wrap, turnkey, internally developed, …) and the transition target (single vs. multiple, homogeneous vs. heterogeneous, …).

Provide a subsection for each stakeholder category with a significant transition role. Besides developers, maintainers, users, customers, and interfacers, these could include operational testers, labor unions, and safety officials for example.

Where the LCP focuses on organizational roles, this section should focus on the roles of specific individuals (e.g., which maintainer individuals are going to perform representative product modifications to evaluate maintainability?). These commitments need to be coordinated with the stakeholder organizations and individuals.

4.  Milestone Plan

The plan should include key preparation milestones leading up to the key Transition Readiness Review (TRR) milestone, and key transition milestones leading up to the Product Release milestone. There may be post-release milestones as well: for example, if user training is spread across time.
Any explicit or implicit client milestone commitments need to be coordinated with the clients. Implicit commitments can arise from dependency relations, e.g., data preparation → training material preparation → training → operational use → operational evaluation.

577b Guidelines:

All developer activities are assumed to be complete at the Product Release milestone. The schedules and milestones should be compatible with those in the Life Cycle Plan.

5. Required Resources

This section should elaborate the transition portion of LCP 5 and FRD 2.1.2. Resources should include the time required by developers and clients, and the necessary financial resources for hardware, software, facilities, supplies, data, training, and other possible needs.
Software User's Manual

A. Description

1. Purpose

The purpose is to teach and guide the user how to use the product, i.e., the steps for running the software, describes the expected output(s), and describes the measures to be taken if errors occur.

2. Timing

Produce the manuals early in the development process and update through each iteration. This helps users evaluate each release, and ensures early and continuous user feedback. How early in the development cycle to begin producing the user manual depends on the type of system. Systems with complex interfaces or with a lot of user interaction will require early versions of the user manual and also early prototypes of the interface. Embedded systems with little human interface will probably not require an early start on user documentation.

3. Intended Audience

4. Participants

5. Responsibility

The Test Team or the Technical Writer is responsible for creating and updating the material. The user manual can be written by technical writers, with input from developers, or it can be written by the test team, whose members are likely to understand the user's perspective.

6. Completion Criteria

7. Additional Information

The end-user documentation gives instructions for using the software. Provide documentation for all types of users. You may want to consider a different user manual for each operational role or user class: e.g., User Manual, Administrator Manual. Use cases are the basis for the manual.

Use screen shots extensively.

A reason for allocating the user manual to the test team is that it can be generated in parallel with development and evolved early as a tangible and relevant perspective of evaluation criteria. Errors and poor solutions in the user interface and use-case model can be spotted and corrected during the early iterations of the project, when changes are cheaper.

By writing the user manual, the testers will get to know the system well before they start any full-scale testing. Furthermore, it provides a necessary basis for test plans and test cases, and for construction of automated test suites.
8. High-Level Dependencies

The following table shows the dependencies of the Software User’s Manual on other MBASE documents.

<table>
<thead>
<tr>
<th>OCD</th>
<th>SSRD</th>
<th>SSAD</th>
<th>LCP</th>
<th>FRD</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>~</td>
<td>-</td>
<td>~</td>
<td>+</td>
</tr>
</tbody>
</table>

B. Document Sections

1. Introduction

1.1 System Overview

State the purpose of the library system and the software to which this manual applies.

1.2 System Requirements

Describe the minimum hardware and software (Operating System, etc.) requirements for the system.

2. Operational Procedures

Briefly describe functions available to the user and describe the step-by-step procedures for accessing these functions.

Include the following information, as applicable:

a. Initialization. Describe any procedures needed to initialize the software. Identify any initialization options available to the user.

b. User Functions. Describe the functions available to the user. For each function describe:
   1. User inputs including format, limitations, input method, etc.
   2. Operational results

c. System Inputs. Describe the system inputs to the software that may occur while the software is in use and that may affect the interface with the user. Include format, frequency, allowable range, and units of measure, as applicable.

d. Termination. Describe how to terminate software operation and how the user determines that normal termination has occurred.

e. Restart. Describe the procedures for restarting the software.

You may want to split the description of the user manual into:

- Basic Features (Getting Started, or Learning the Basics)
- Advanced Features
3. Installation Procedures

In a system where the end user is expected to install the product, the Installation Instructions can be included in the user's guide. For a more complicated installation where qualified service staff are needed, the installation instructions would be described in a separate Installation Manual.

3.1 Initialization Procedures

Describe first-time installation procedures

3.2 Re-installation

Describe procedures for reinstalling the system (e.g., to recover from a corrupt installation)

3.3 De-installation

Describe procedures for removing the system

4. Troubleshooting

4.1 Frequently Asked Questions

List Frequently Asked Questions by operators, and answers to those questions.

4.2 Error Codes and Messages

List and identify all error codes and messages generated by the software, the meaning of each message, and the action to be taken when each message appears.

5. Notes

Include any general information that aids in the understanding of the document. All acronyms, abbreviations, and their meaning as used in this document should be listed.

6. Appendix

List any additional information, such as technical specifications, etc.
System and Software Support Plan (SSSP)

A. Description

1. Purpose

The purpose of the Support Plan is to guide the system’s support stakeholders (administrators, operators, maintainers) in successfully operating and maintaining the system.

2. Timing

The support plan builds on the Operational Concept Description and the Support Stage elements of the Life Cycle Plan. It is developed during the Construction Phase and updated during the Transition Phase.

3. Intended Audience

The support stakeholders are the primary audience for the Support Plan. The developers use it as a target for the Transition Plan. The users rely on it for operational information.

4. Participants

5. Responsibilities

The Support Plan is the joint responsibility of the support stakeholders and the developers. Normally, the support stakeholders have the lead responsibility.

6. Completion Criteria

Stakeholder concurrence, feasibility of execution, and realism with respect to available resources. These are evaluated by the equivalent of an Architecture Review Board at the Transition Readiness Review.

7. Additional Information

577b Guidelines:

This version of the Support Plan is developed by the 577b team as a starting point for the support stakeholders’ more detailed Support Plan. Normally, support plans can assume some post-IOC availability of the developers, but this is not the case for 577b students.

8. High-Level Dependencies

The following table shows the dependencies of the System Software Support Plan on other MBASE documents.
B. Document Sections

1. Support Objectives and Assumptions

1.1 Support Objectives

Identify the key driving objectives for software support. Some possible examples are:

- The top priority objective is to ensure [system safety; Platinum customer satisfaction; speed of competitive reaction in the marketplace].
- Replace inefficient legacy systems as quickly as possible.
- Provide more promising and stimulating career paths for support personnel.

1.2 Support

State any assumptions which, if invalid, would make the support plan unworkable. Some possible examples are:

- Continuity of funding, staffing, and upper management support.
- Controllable/negotiable interfaces with interoperating systems.
- Stability of requirements and schedules for each release.

2. Support Strategy and Environment

2.1 Support Strategy

2.1.1 Support Lifetime

Provide a best estimate of the envisioned support lifetime. It is acceptable to provide approximate estimates (less than one year; over 5 years) or relative estimates (until the current Master Status package is phased out).

2.1.2 Release Strategy

Identify the overall release strategy (continuous as-needed small fixes; major annual releases and as-needed small fixes; releases synchronized with major COTS upgrades, etc.). Identify any known content of early releases (evolution requirements in SSRD 6). Describe the transition strategy for new releases (alpha and beta testing; pilot operation; sequencing of multisite or multinational releases).

2.1.3 Release Requirements Determination

Identify the primary drivers of new release content (budget, schedule, staffing, legal, business opportunity). Describe the process by which release requirements are determined (executive prioritization; stakeholder win-win; bidding; organizational sub–allocations, etc.).
2.1.4 Release Process

Describe the process for each release and how its phases overlap with neighboring releases. Normally, each release should have an Inception, Elaboration, Construction, and Transition phase, with associated intermediate LCO, LCA, TRR/IOC, and RRR milestones. Identify the incremental strategy for the Construction phase.

2.2 Support Environment

Document the anticipated environmental concerns for the long-term use of the system. These should be consistent with Section 2 of the Transition Plan, but more complete with respect to multisite environment variations. Reference the Production Stage Deliverables in LCP 2.3 where appropriate.

2.2.1 Hardware

Describe the hardware and associated documentation needed to maintain the deliverable software. This hardware may include computers, peripheral equipment, and non-computer equipment.

2.2.2 Software

Identify and describe the software and associated documentation needed to maintain the deliverable software. This software may include computer-aided software engineering (CASE) tools, compilers, test tools, test data, utilities, configuration management tools, databases and other software. The description shall include:

a. Specific names, identification numbers, version numbers, release numbers, and configurations, as applicable
b. Rationale for the selected software
c. Reference to user/operator manuals or instructions for each item, as applicable
d. If items must be acquired, information about a current source of supply, including whether the item is currently available and whether it is expected to be available at the time of delivery
e. Information about vendor support, licensing, and data rights, including whether the item is currently supported by the vendor, whether it is expected to be supported at the time of delivery, whether licenses will be assigned to the support agency, and the terms of such licenses
f. Security and privacy considerations, limitations, or other items of interest

577b Guidelines:

Include any special settings for various tools (such as compilers, database management systems), environment variables that the maintainer needs to be aware of to properly maintain the system.

2.2.3 Facilities

Describe the facilities needed to maintain the deliverable software. These facilities may include special building features, such as cabling, special power requirements, etc.

2.2.4 Other documentation

This section should supplement the Project Deliverables in LCP 2.3 with the following information:
a. Identification of additional documents as acquirer-furnished, an item that will be delivered to the support agency, an item the support agency is known to have, an item the support agency must acquire, or other description of status.

b. If a document must be acquired, information about where to acquire it

c. Information about licensing and data rights, in particular license expiration dates

d. Security and privacy considerations, limitations, or other items of interest

3. Support Responsibilities

Identify the major support stakeholder roles, and the organizations and individuals (where possible) responsible for each. Roles may include:

a. Software maintenance (corrective, adaptive, perfective)

b. System administration (version control, backup/recovery, authorizations, performance management, etc.)

c. Operational and user support (help desks, training, etc.)

d. Database administration

e. Data acquisition

Include anticipated number of personnel, types and levels of skills and expertise. This paragraph shall cite, as applicable, actual staffing on the development project as a basis for the staffing needs cited.

Describe any special responsibility parameters (e.g., duty cycle: 24x7; Mon-Fri 8am-5pm; on-call; etc.); and special arrangements for shared or backup responsibilities.

This section should derive from and be consistent with LCP Section 3.

4. Support Approach

Identify differences between the support approach and the development approach in LCP Section 4:

4.1 Monitoring and Control

4.2 Methods, Tools, and Facilities

4.3 Configuration Management

4.4 Quality Management

Include advice and lessons learned that the developer may wish to recommend to the support agency for supporting the deliverable software and associated support environment, such as:

- Regression Test Cases
- COTS upgrade guidance
- Tools and Procedures
5. Support Resources

5.1 Support Budgets

Include a support Work Breakdown Structure and at least initial budgets for the WBS elements. The budgets should be consistent with the cost estimates and business case in FRD 2.1.

5.2 Other resources

This paragraph shall identify any other resources needed to support the deliverable software. Included may be consumables such as magnetic tapes and diskettes, together with an estimate of the type and number that should be acquired.
Guidelines for MBASE

Sources and References

The guidelines have drawn upon material from the following:

A. Overall


Since the guidelines have been integrated from multiple sources, they will not necessarily be fully consistent with the guidelines in any individual source. See the following resources for useful perspectives and additional guidelines.

B. Operational Concept Description


C. System and Software Requirements

Definition


Template. James & Suzanne Robertson. Atlantic Systems Guild
http://www.atlsysguild.com/Site/Robs/Tempsect.html

Typical list of data items for system/software development
http://www.airtime.co.uk/users/wysyg/widlist.htm

D. System and Software Architecture

Description

C2 Architectural style (see http://www.ics.uci.edu/pub/arch/c2.html)


E. Life Cycle Plan


Appendices

A. Suggested Win–Win Taxonomy for MBASE
B. Level of Service Requirements
C. Common Definition Language (CDL) For MBASE
Appendix A. Suggested Win–Win Taxonomy for MBASE

The suggested domain taxonomy to be used as a checklist and organizing structure for the Win–Win requirements negotiation. Each Win–Win stakeholder artifact should point to at least one taxonomy element (modify taxonomy as appropriate). Each taxonomy element should be considered as a source of potential stakeholder win conditions and agreements. The Win–Win taxonomy roughly corresponds to the table of contents of the System and Software Requirements Definition (SSRD). Mapping the Win–Win taxonomy to the SSRD outline is straightforward, but in some cases, some sections need to be combined. In particular, Operational Modes are described in the SSRD with System Requirements. The reason is that the same system functionality may lead to different results depending on the mode.

1. Project Constraints (===>SSRD 2. Project Requirements)
   1.1. Budget Constraints
   1.2. Schedule Constraints
   1.3. Staffing Constraints

2. Application Capabilities (===>SSRD 3.2 System Requirements)
   2.1. Operational Modes
   2.2. User Classes
   2.3. Mission Capabilities. These will vary depending on whether the mission involves a multimedia archive, selective dissemination of information, data analysis, etc.
   2.4. Support Capabilities
      2.4.1. Help
      2.4.2. Administration
         2.4.2.1. User Account Management
         2.4.2.2. Usage Monitoring and Analysis
      2.4.3. Maintenance and Diagnostics

3. Level of Services (===> SSRD 5. Level of Service Requirements)
   3.1. General Qualities
      3.1.1. Correctness
      3.1.2. Simplicity
      3.1.3. Consistency
      3.1.4. Completeness
   3.2. Coherence
   3.3. Dependability
      3.3.1. Reliability
      3.3.2. Accuracy
      3.3.3. Availability
      3.3.4. Survivability
      3.3.5. Serviceability
      3.3.6. Verifiability
      3.3.7. Resilience
   3.4. Security
      3.4.1. Integrity
      3.4.2. Privacy
      3.4.3. Audit
   3.5. Confidentiality
   3.6. Safety
   3.7. Interoperability
      3.7.1. Compatibility
   3.8. Usability
      3.8.1. Mission Orientation
      3.8.2. Comprehensiveness
3.8.3. Controllability
3.8.4. Ease of Learning
3.8.5. Ease of Use
3.8.6. Help requirements

3.9. Performance
3.9.1. Processing Efficiency
3.9.2. Memory Efficiency
3.9.3. Storage Efficiency
3.9.4. Network Efficiency

3.10. Adaptability (Evolve-ability)
3.10.1. Portability
3.10.2. Flexibility
3.10.3. Scalability/Expandability/Extendability/Extensibility
3.10.4. Modifiability
3.10.5. Maintainability
3.10.6. Reconfigurability

3.11. Reusability

4. Interfaces (SSRD 4. System Interface Requirements)
4.1. User Interfaces Requirements
4.1.1. Graphical User Interfaces
4.1.2. Command-Line Interfaces
4.1.3. Application Programming Interfaces
4.1.4. Diagnostics
4.2. Hardware Interfaces
4.3. Communications Interfaces
4.4. Other Software Interfaces

5. Environment and Data (SSRD 2. Project Requirements)
5.1. Design and Construction Constraints
5.1.1. Tools
5.1.2. Programming Languages
5.1.3. Computer Resources
5.1.4. Standards Compliance
5.2. Packaging
5.3. Implementation
5.4. Software Support Environment Requirements

6.1. Capability Evolution
6.2. Interface Evolution
6.3. Technology Evolution
6.4. Environment Evolution
6.5. Workload Evolution
Appendix B. Level of Service Requirements


Accuracy

(1) A qualitative assessment of correctness, or freedom from error; (2) A quantitative measure of the magnitude of error

Adaptability

Adaptability is defined by the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed. Syn: Flexibility

Audit

Specification of the required audit checks or various audit trails the system should keep to build a system that complies with the appropriate audit rules. This section may have legal implications

Availability

The degree to which a system or component is operational and accessible when required for use. Often expressed as a probability. See also: Error Tolerance; Fault-tolerance; Robustness

Compatibility

(1) The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment (2) The ability of two or more systems or components to exchange information (See also: Interoperability)

Complexity

(1) The degree to which a system or component has a design or implementation that is difficult to understand and verify. Contrast with: simplicity

Consistency

The degree of uniformity, standardization, and freedom from contradiction among the documents or parts of a system or component.

Correctness

Correctness is defined by: (1) The degree to which a system or component is free from faults in its specification, design, and implementation; (2) The degree to which software, documentation, or other items meet specified requirements; (3) The degree to which software, documentation, or other items meet user needs and expectations, whether specified or not.

Dependability

Dependability is defined as “that property of a computer system such that reliance can justifiably be placed on the service it delivers” [Laprie, 1992]. Depending on the intended application of the system dependability is usually expressed as a number of inter-dependent properties such as reliability, maintainability and safety. It refers to a broad notion of what has historically been referred to as “fault tolerance”, “reliability”, or “robustness”
Guidelines for MBASE

Appendix B. Level of Service Requirements

Efficiency

Efficiency is defined by the degree to which a system or component performs its designated functions with minimum consumption of resources.

Error Tolerance

The ability of a system or component to continue normal operation despite the presence of erroneous inputs. See: Fault tolerance, robustness

Expandability/Extendability/Extensibility

Expandability is defined by the easy with which a system or component can be modified to increase its storage or functional capability.

Fault-tolerance

(1) The ability of a system or component to continue during normal operation despite the presence of hardware or software faults. See also: error tolerance; fail safe; fail soft; fault secure; robustness

Flexibility

Flexibility is defined by the easy with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

Integrity

Integrity is defined by the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data.

Example: "Identical up-to-date booking information must be available to all users of the system."

Interoperability

Interoperability is defined by the ability of two or more systems or components to exchange information and to use the information that has been exchanged.

Describe other platforms or environments on which the system is expected to run without recompilation.

Example: "The program should be binary compatible with Windows 3.1, Windows 95 and Windows 98"

Legality

Describe any legal requirements for this system, to comply with the law to avoid later delays, lawsuits and legal fees.

If the legal requirements are above average, then this section might need to be entirely revisited

Example: "Personal information must be implemented so as to comply with the data protection act."

Example: "The system shall not use any image formats that might infringe with existing copyrights or pending legislation (e.g., GIF)"

Maintainability

Maintainability is defined by: (1) The easy with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment; (2) The easy with which a hardware system or component can be retained in, or restored to, a state in which it can perform its required functions.
Guidelines for MBASE

Guidelines for MBASE Appendix B. Level of Service Requirements

Memory Efficiency

List of memory usage requirements that have a genuine effect on the system's ability to fit into the intended environment to set the client and user expectations.

Example: "The system should be able to run on a multi-tasking system with 4MB of free memory"

Example: "Upon exit, the server shall return all the memory it allocates to the pool of free memory on the host computer without any memory leaks".

Modularity

The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

Network Efficiency

List of network usage requirements that have a genuine effect on the system's ability to fit into the intended environment to set the client and user expectations.

Example: "The system should not increase the traffic on the current network by more than 10%"

Performance

Performance is defined by the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage.

Political Correctness

Describe any special factors about the product are necessary for some political or socioeconomic reason: the reality is that the system has to comply with political requirements even if you can find a better/more efficient/more economical solution.

Example: "Our company policy says that we must buy our hardware from Unisys."

Portability

The ease with which a system or component can be transferred from one hardware or software environment to another.

- Describe other platforms or environments to which the system must be ported to quantify client and user expectations about the platforms and future environments in which the system is expected to run.

- Example: "The source code should compile correctly on Solaris and Linux"

Privacy/Confidentiality

Specification of who has authorized access to the system, and under what circumstances that access is granted.

Processing Efficiency

List of response time requirements that have a genuine effect on the system's ability to fit into the intended environment to set the client and user expectations for the response time of the system.

Reliability

Reliability is defined by the ability of a system or component to perform its required functions under stated conditions for a specified period of time.
Reusability

Reusability is the degree to which a software module or other work product can be used in more than one computer program or software system.

Software reusability means that ideas and code are developed once, and then used to solve many software problems, thus enhancing productivity, reliability and quality. Reuse applies not only to source-code fragments, but to all the intermediate work products generated during software development, including requirements' documentation, system specifications, design structures and any information the developer needs to create software.

Robustness

The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions. See also: error tolerance; fault tolerance

Scalability

A quantification of how the system should be able to adapt to an increase in the workload without imposing additional overhead or administrative burden.

Storage Efficiency

The degree to which a system or component performs its designated functions with minimum consumption of available storage.

Example: "The system should be able to run with only 40MB of available disk space"

Usability

Usability is defined by the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

Ease of Learning

A statement of the expected learning time, and any special training needed, for the expected users of the system to guide the designers of the system's interface and functionality and to determine whether or not the user can use the system after the number of hours training/familiarization/use (plus description of training program if applicable) for each type of user.

Example: "The average user should be able to produce a virtual tour within 1 hour of beginning to use the system, without resorting to the manual."

Make sure that you have considered the ease of learning requirements from the perspective of all the different types of users.

Ease of Use

A statement of how easy the system is to use to guide the system's designers.

Example: "The average user must have an error rate less than 2%.

Make sure that you have considered the usability requirements from the perspective of all the different types of users. It is necessary to make some measure of the system's intended usability. This may be that the user labs pronounce that the system is usable, or that it follows all the Apple/Windows interface guidelines, or simply that it must be popular with the majority of users.

Help requirements
A description of the help that the system will provide. The help requirements might become so complex that it is better to treat help as a separately specified system.

Example: "The system must provide context specific help. The user must be able to select an artifact and receive instruction about its use."

These might be requirements that relate to all events (globally accessible help facilities) or they might be requirements that relate to individual events or functional requirements.
Appendix C. COMMON DEFINITION LANGUAGE (CDL) for MBASE

Abbreviations: A: Analysis  DD: Domain Description  D: Design

Abstraction DD: A simple interface on complex information. An abstraction is a representation of something, either tangible or conceptual.

Algorithm D: That portion of a behavior that actually carries out the work of the behavior, independent of any decision-making (policy making) necessary to determine which algorithm to execute.

Analogy: The identification of groups of similar relationships between abstractions. An operation abstraction.

Analysis: The part of the software development process whose primary purpose is to formulate a model of the problem domain. Analysis focuses on what to do, design focuses on how to do it. See design.

ARB: Architectural Review Board

Architect: The person or persons responsible for evolving and maintaining the system’s architecture. Ultimately, the architect gives the system its conceptual integrity.

Architecture: A description of the organization and structure of a system. Many different levels of architectures are involved in developing software systems, from physical hardware architectures to the logical architecture of an application framework.

Describes the static organization of software into subsystems interconnected through interfaces and defines at a significant level how nodes executing those software subsystems interact with each other.

Audience: The person or persons who act as the consumers of an abstraction’s interface.

Behavior Model: A representation of the behaviors in a system.

Behavior: Maps to objects.

Boundaries of Control: The point at which a behavior requires interaction with users or other elements outside the system.

Classification: Organizing a set of abstractions according to their common qualities. Classification allows you to reduce complexity in an object model. By making general statements about groups of objects.

CLI: Command Line Interface

CM: Configuration Management

Coherence: A measure of an abstraction’s elegance. An abstraction is coherent if all of its quality resolutions are both correct and consistent.

Cohesiveness: A measure of an abstraction’s elegance. The degree to which an abstraction’s qualities fit with the defining quality and with each other.

Common Definitional Language (CDL): A glossary. A common and consistent set of problem space and solution space terminology developed during modeling. Used as a catalog and thesaurus during systematic reuse domain engineering to describe the key concepts.

Comparator: A similarity between two abstractions. If the similarity between the two abstractions is expressed in terms of each abstraction’s defining quality, the Comparator is referred to as the Main Comparator. Contrast with Discriminator.

Completeness: A measure of elegance describing whether all of an abstraction’s information can be accessed from the interface.

Component: A meta-type whose instances are ‘part-of’ another abstraction. Components are needed to describe the system to domain experts. Components are compositions of objects (sometimes only one). What an entity is in the domain description, a component is in the system analysis. Components are nouns.

Composition: Creating a new abstraction by combining smaller components into a larger abstraction. Contrast with Decomposition. An operation on an abstraction.
Conceptualization: The earliest phase of development, focused upon providing a proof of concept for the system and characterized by a largely unrestrained activity directed toward the delivery of a prototype whose goals and schedules are clearly defined.

Constraint: A restriction on the resolution of a component’s or an object’s quality. For instance, an attribute might have a minimum value, or a set of illegal values. An attribute quality

Context: The surrounding environment of a software project.

COTS: Commercial Off The Shelf

Coverage: A measure of elegance, with regard to an object’s defining quality. A defining quality has good coverage if it includes everything that should be a part of the abstraction.

CTS: Construction Transition Support package

Customer: Customers request application systems, place requirements on them, and usually pay for the systems. Customers also interact when deciding on needed features, priorities, and roll-out plans when developing new versions of component systems and the layered system as a whole. Customers can be both internal, such as business process owner, or external, such as another company.

Decomposition: Breaking an abstraction into smaller components. An operation on an abstraction.

Dependency: A requirement that specifies how one element of a system affects another. There are three possible dependencies. Dependencies can be Semantic – how a quality of an abstraction( object, attribute, behavior, relationship) is resolved under a given set of conditions, e.g. vacation depends on start date; Functional – describes how a component uses other components to assist in providing behavior, e.g. alarm – response; or Conceptual – effects the defining of qualities, e.g. what a car is may depend on what model it is. Dependency is an attribute quality.

Design: The part of the software development process whose primary purpose is to decide how the system will be implemented. During design, strategic and tactical decisions are made to meet the required functional and quality requirements of a system. See analysis. Discriminator: A difference between two abstractions. If the difference between the two abstractions is expressed in terms of each abstraction’s defining quality, the discriminator is referred to as the Main Discriminator.

Domain: The part of the real world that a particular software project is intended to model. Compare with Context.

Domain Expert: A person very familiar with the subject matter of the domain and how it fits together.

Domain Model: The sea of classes in a system that serve to capture the vocabulary of the problem space; also known as a conceptual model. A domain model can often be expressed in a set of class diagrams whose purpose is to visualize all of the central classes responsible for the essential behavior of the system, together with a specification of the distribution of roles and responsibilities among such classes.

Elegance: An abstraction that conveys its underlying information using the simplest possible interface. A measure of an abstraction’s elegance is its Information-to-interface Ratio. Qualities that directly impact elegance are: Completeness, Cohesiveness. Sufficiency, and Coherence.

Encapsulation: A property of object-oriented programs in which an object’s implementation is hidden behind its interface.

Engineering: Creating cost-effective solutions to practical problems by applying scientific knowledge to building things of value.

Engineering Abstractions: Construction of elegant abstractions to increase the information/interface.

Enterprise Classification Model: The complete model of a domain, including object structures and behaviors.

Entity (Organization): Any identifiable set of individuals, policies or systems.

Entities Model: Entities are the fundamental building blocks of the Domain Description that represent information stored in the system.

Factoring: Identifying common elements of two or more abstractions, typically decomposition followed by composition. An operation abstraction.


Framework: A collection of interdependent classes that provides a set of services for a particular domain;
a framework thus exports a number of individual classes and mechanisms that clients can use or adapt.

**FRD:** Feasibility Rationale Description

**Functional:** See Functional Dependency under Dependency.

**Generalization:** Creating a more inclusive classification. Within an abstraction hierarchy, generalization results in “kind of” relationships. Contrast with Specialization. An operation on an abstraction. There are three special cases of generalization: Leading Special Case: easy to handle and very accessible in which it is seen that other cases follow; Representative Special Case: is a specialization achieved by resolving some of the abstraction’s qualities in an arbitrary way; Extreme Special Case: sets boundaries for other cases.

**Goal:** motivation neither expressed nor implied by responsibilities. From notes, “factors that contribute to the choices and aspirations of the organization.”

**Hierarchy:** A directed tree of abstractions with transitive relationships between levels.

**Identity:** Designation of a component such as a name or phone number. An attribute quality.

**Item Qualification Testing:** See Software Item Qualification Testing.

**Information:** Processed data that conveys more than the data itself; relationships or descriptions of data.

**Input:** An operation quality. Any data that is required to carry out the operation.

**Interaction:** mutual or reciprocal action or influence between entities and/or systems.

**Interface:** Set of qualities of an object/entity that may be extracted or changed. Refers to that part of an object/entity which is accessible to others.

**Law of Demeter:** A hierarchy of operations with respect to where messages should be sent, e.g. first to itself.

**Meta Level:** Abstractions that describe other abstractions.

**Metric:** Measures. E.g. elegance metrics such as cohesiveness, consistency etc.

**Model:** An organized collection of abstraction levels.

**Object:** An encapsulated packet of data and a behavior that acts as abstraction of a particular domain element or programming construct.

**OCD:** Operational Concept Description

**Operation:** A task which is executed in response to the stimulus of an event. The functionality of a component. Involve data flow, control flow and state diagrams. Map to abstractions (see behavior).

**Operations Engineering:** Facilitates ways of organizing and managing complex operations, e.g. through assembling operations into hierarchies.

**Operations Qualities:** Trigger, Scenario, Preconditions, Postconditions, Inputs, Outputs, Actions, Exceptions.

**Operations Classification:** Organizing operations according to abstractions.

**Output:** An operation quality. Any data that is produced by the operation.

**Participating Agent:** Non developer stakeholder in the system

**Performing Agent:** One that expends effort to realize the solution to the problem being solved in the project (manager, architect, analyst, designer)

**Policy:** That portion of a given behavior that decided what the behavior should be doing. Contrast with algorithm.

**Precision:** A measure of elegance, with regard to an object’s defining quality. A defining quality is precise it is excludes everything that is not part of the abstraction.

**Precondition:** An operation quality. A prerequisite set of conditions that must be true in order for an operation to proceed,

**Primitive Method:** A method that directly accesses an instance variable.

**Readability:** Visibility of the value. All attributes should be readable. A quality of an attribute. A measure of accessibility

**Reflexivity:** Indicates whether a relationship can have the same object at both ends. Reflexive relationships can; irreflexive relationships cannot.

**Regression Testing:** Selective testing of a system or component to verify that modifications have not
caused unintended effects and that the system or component still complies with its specified requirements.

**Relationship:** A conceptual connection between two or more components or objects. A complex relationship is a composition of simple relationships and include bidirectional relationships (simple “one to many” relationships) and symmetric relationships (a relationship that has the same qualities when viewed from either direction (e.g. “next to”)).

**Relationship Constraints:** Three of them (Reflexivity – relationships that have the same components as both the source and destination e.g. lawyers as own clients; Directed – limits the possible relationships between two components to one or more relationships e.g. selected from; Mappings – the existence of a relationship requires the existence of another – e.g. ‘works for’ implies ‘employed by’

**Relationship Types:** ‘Part of’ relationship (the relationship objects within a component have to their container).

‘Contains’ relationship is the reverse of the ‘part’ relationship (deletion of the container deletes the parts). ‘Composite’ relationships, four of them: (Collections – e.g. address book; Aggregations – e.g. an automobile engine with its aggregation of parts; Groupings – like an aggregation but if you delete all the parts the group is deleted as well; Partnerships – where the deletion of a relationship between objects causes the container to be deleted)

**Relevance:** The degree to which a quality is important in the current context.

**Responsibility:** System responsibilities are a list of tasks the final system will be responsible for. Something to be done.

**Reuse:** Further use or repeated use of an artifact. Typically software programs that were designed for use outside their original context.

**RLCA:** Rebaselined LCA

**ROI:** Return on Investment

**Scenario:** An operation quality. A description of the steps taken to complete the operation.

**Software Item:** An aggregation of software, such as a computer program or database, that satisfies an end use function and is designated for purposes of specification, qualification testing, interfacing, configuration management, or other purposes.

Software items are selected based on trade-offs among software function, size, host or target computers, developer, support strategy, plans for reuse, criticality, interface considerations, need to be separately documented and controlled, and other factors. A software item is made up of one or more software units.

**Software item qualification testing:** The developer shall perform software item qualification testing in accordance with the following requirements.

**NOTES:**
1-Software item qualification testing is performed to demonstrate to the acquirer that software item requirements have been met. It covers the software item and interface requirements in System and Software Requirements Definition (SSRD). This testing contrasts with developer-internal software item testing, performed as the final stage of unit integration and testing.
2-If a software item is developed in multiple builds, its software item qualification testing will not be completed until the final build for that software item, or possibly until later builds involving items with which the software item is required to interface.
Software item qualification testing in each build is interpreted to mean planning and performing tests of the current build of each software item to ensure that the software item requirements to be implemented in that build have been met.

**Software Unit:** An element in the design of a software item; for example, a major subdivision of a software item, a component of that subdivision, a class, object, module, function, routine, or database. Software Units may occur at different levels of a hierarchy and may consist of other software units. Software units in the design may or may not have a one-to-one relationship with the code and data entries (routines, procedures, databases, data-files, etc.) that implement them or with the computer files containing those entities.

**Subsystem:** A model element which has the semantics of a package, such that it can contain other model elements, and a class, such that it has behavior. (The behavior of the subsystem is provided by classes or other subsystems it contains). A subsystem realizes one or more interfaces, which define the behavior it can perform.

**Sufficiency:** A measure of an abstraction’s elegance. The degree to which all of an abstraction’s information can be accessed in a reasonable amount
of time, or with a reasonable amount of knowledge about the domain.

Super-type: Similar to generalization. Creating a less-specific, more inclusive class from a set of subclasses

System: As an instance, an executable configuration of a software application or software application family; the execution is done on a hardware platform. As a class, a particular software application or software application family that can be configured and installed on a hardware platform. In a general sense, an arbitrary system instance.

System qualification testing: The developer shall participate in system qualification testing in accordance with the following requirements.

NOTES:
1 – System qualification testing is performed to demonstrate to the acquirer that system requirements have been met. It covers the system requirements in the System/Subsystem Specifications (SSE) and in associated Interface Requirements Specifications (IRSs). This testing contrasts with developer-internal system testing, performed as the final stage of software/hardware item integration and testing.
2 – If a system is developed in multiple builds, qualification testing of the completed system will not occur until the final build. System qualification testing in each build is interpreted to mean planning and performing tests of the current build of the system to ensure that the system requirements to be implemented in that build have been met.

Test case: A set of test inputs, execution conditions, and expected results developed for particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. [NOTE: not the same definition as used in IEEE J-016.]

Scope: The value of an attribute and whether or not another component of the same type can have the same value. A quality of an attribute.

SDP: Software Development Plan

Selector: A relational attribute which uniquely identifies an object in the context of the relationship to which it belongs.

Semantic: Semantic equivalence is one component simply representing the state of another component.

Appendix D. Status Reports

E.g., bank account can be represented by open account.

Source Component: The component that originates the relationship. (See also Destination component).

Source: Specialization: Refining a classification by adding more qualities to it. Contrast with Generalization.

Specialization: Refining a classification by adding additional qualities to it (sub class). An operation on an abstraction.

Spiral: A model of a system development which repeatedly cycles from function to form, build, test, and back to function.

SSRD: System and Software Requirements Definition

SSAD: System and Software Architecture Description

State: A combination of attributes and relationships that affects the behavior of an object and has well-defined transitions to or from other discreet states. E.g. solvency of a bank account. State may be represented by an attribute.

Test class: An optional (only used if designing and implementing test specific functionality) stereotype of Class in the design model.

Test model: A collection of test cases and test procedures and their relationship. A test case can be implemented by one or more test procedures, and a test procedure may implement (the whole or parts of) one or more test cases.

Test procedure: A set of detailed instructions for the set-up, execution, and evaluation of results for a given test case (or set of test cases).

Transitive (Transitivity): The relationship: If A then B, If B then C. Therefore if A then C.

Trigger: An operation quality. A set of conditions which, when true, cause an event to be sent to stimulate an operation.

Types: Types of components in the same class must share all qualities of other components in that class. E.g. names.

UTCR: Unit Test Completion Review
**Guidelines for MBASE**

**Value Class:** A class that is used to represent an atomic value, such as a string or a number.

**Appendix D. Status Reports**

**Waterfall:** A development model based on a single sequence of steps
Appendix D. Status Reports

The CS577 [weekly] status reports are shown in this appendix.
# Weekly Effort Form

**Week #**

<table>
<thead>
<tr>
<th>Name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Management</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Life-Cycle Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control and Monitoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client Interaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team Interaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Web Site</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARB Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Environment and Configuration Management</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training and Preparation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Installation and Administration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom Toolsmithing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COTS Initial Filtering (Assessment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Win–Win Negotiation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping for Requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modeling for Operational Concept Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documenting of Operational Concept Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modeling for System and Software Requirements Definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documenting of System and Software Requirements Definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modeling for System and Software Architecture Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documenting of System and Software Architecture Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COTS Final Selection (Evaluation)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COTS Tailoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component Prototyping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Critical Component Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COTS Glue Code Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Assessment</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Case Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feasibility Rational Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inspections and Peer Reviews</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Deployment</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transition and Support Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Other Effort Not Covered Here</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Weekly Status Report**

_**Team #**_

_**Team Members**_

_**Project Title**_

_**Week #**_

This is a suggested Status Report. Feel free to expand or collapse the various categories.

**Progress**

- Describe the progress on the project during the past week, both in qualitative and quantitative terms
- List any significant accomplishments since the last status report. This can include the completion/revision of project deliverables
- Include metrics to describe the progress
- Attach a copy of the major documents' and code change history (easily provided from configuration management system)
- If problem reports and/or defect/issue lists were exchanged electronically, either within the team or with the IV&V person, please provide a list of the message subject and date

**Problems**

- List any problems the team is experiencing. This includes things that have caused delays, or any other roadblocks to anticipated progress.
- Provide a weekly Top-N risk items list as follows:

<table>
<thead>
<tr>
<th>Risk Items</th>
<th>Weekly Ranking</th>
<th>Risk Resolution Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current</td>
<td>Previous</td>
</tr>
</tbody>
</table>
• Provide a weekly # of open problem report and # of issues list as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>New(^{17}) opened</th>
<th>New(^{17}) closed</th>
<th>Old(^{18}) closed</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issues(^{19})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defects(^{20})</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem Reports</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Immediate Plans**

List your team's plans (or TO DO items) for the next week. This section might be filled with significant action items that are mentioned during your team status meeting, to help remind team members of significant items discussed in the meeting. You are encouraged to maintain, separately, meeting minutes, to record important discussions and action items raised during your team meetings.

**Comments**

Include here any items not covered above.

---

\(^{17}\) Problem reports and issues generated since last report.

\(^{18}\) Old are those in "Current" in the previous report.

\(^{19}\) See QMP for a definition. Usually potential defects identified in other artifacts during a review (i.e., artifacts not the primary focus of the review)

\(^{20}\) See QMP for a definition. Usually changes made by the "author" of a reviewed artifact as a result of an "issue" or those identified in an inspection (where a "Defect" is objectively defined and the author concurs with the designation.)
Guidelines for MBASE

Index


Actors ...............................................................24, 77, 79, 104, 114, 123

Automated Analysis ...........................................261, 262, 263

Budget ....................................................54, 206, 207, 213, 221, 235, 314

Business Case .................................................7, 9, 15, 226, 238, 328

Capabilities Implemented ....................................7, 9, 15, 226, 238, 328

Change Request Reporting ..................................252

Change Summary 11, 52, 70, 197, 226, 259, 270, 274, 284, 295

Choice of architecture .......................................66

Classification Model ...........................................5, 322

Classification of Defects .....................................285

COCOMO 15, 195, 222, 227, 232, 234, 237, 243, 244

Code Generation ..........................................................242

Coding ...........................................................218, 241, 259

Collaboration Diagram 18, 19, 35, 37, 71, 72, 73, 170

Compatibility Notes ..............................................255


Component Model ..................................................1

Configuration management .....................................264

Configuration Model .............................................191, 192

COQUALMO ..................................................261, 297

De-installation ....................................................305

Design Guidelines ..................................................259, 266

Detailed Design ...................................................211, 241

Development Cost .............................................34, 226, 230, 231

Domain Description 7, 9, 19, 21, 25, 26, 31, 33, 35, 37, 39, 41, 47, 65, 71, 193, 321, 322

Engineering Stage ...........................................195, 198, 200, 201, 205

Environment Evolution .......................................315

Evolution requirements .........................................63

Execution Testing ...............................................261

Feasibility Criterion ...........................................66, 67

Feasibility Rational Description ............................328

Framework .....................................................5, 107, 117, 311, 322

Gantt Charts ...................................................222

Hardware Interface .............................................60, 315

Hardware Preparation .........................................275, 299

Identification System .........................................264

IKIWISI ...........................................................47, 261, 287

Inspections ....................................................262, 287, 296, 297, 328

Installation .......................................................55, 202, 222, 276, 305, 328

Intended Audience 9, 195, 224, 247, 251, 254, 257, 269, 273, 283, 294, 298, 303, 306

Interface Evolution .............................................64, 315

Inventory ..........................................................235, 255

Iteration Assessment ...........................................202, 241, 242, 244, 250, 251

Iteration Plan 201, 202, 241, 242, 244, 247, 248, 250, 252, 253

Key Stakeholders ................................................13, 15, 43, 44

Known Bugs ..........................................................256

Level of Service 9, 33, 34, 49, 50, 51, 52, 53, 57, 58, 61, 62, 64, 68, 84, 217, 229, 230, 231, 233, 248, 249, 313, 314, 316

Level of Service Evolution .....................................64
Guidelines for MBASE


Life Cycle Strategy ............................................... 198

Maintenance Cost .................................................. 227

Milestone Plan ....................................................... 301

Open Problems ...................................................... 252

Operational Concept Description ..... 1, 2, 7, 195, 201, 209, 210, 220, 240, 241, 243, 306, 311, 323, 328

Operational Cost .................................................... 227

Organization Charts ................................. 43, 206

Peer Reviews 209, 210, 211, 220, 230, 241, 243, 248, 261, 262, 287, 295, 296, 297, 328

Performance Levels ............................................... 249

Phase Milestones ............................................... 199, 218

Physical Inventory .......................................................... 255

Process Match .......................................................... 197, 200, 233

Process Rational ....................................................... 233

Product Features ....................................................... 238

Product Rational ....................................................... 226

Production Stage... 198, 200, 201, 202, 206, 210, 308

Project Constraints.................................................. 14, 15, 29, 314

Project Deliverables.................................................. 201, 300, 308

Project Goals 8, 9, 28, 51, 53, 54, 57, 58, 68, 228, 233

Project Library .......................................................... 215

Quality Guidelines .......................................................... 259

Quality management .................................................... 215

Release Notes .......................................................... 202

Requirements Management ........................................ 241

Requirements Verification ........................................ 263

Results Chain.. 8, 12, 13, 14, 15, 43, 44, 52, 197, 205, 206, 224, 227

Return on Investment ........................................... 227, 324

Risk Assessment ........................................... 207, 236

Risk Management.. 207, 213, 225, 237, 243, 250, 311

Risk Monitoring ....................................................... 212


Sequence Diagram .................................................... 182, 184

Shared Vision .......................................................... 11, 14, 15, 28

Software Preparation ............................................. 275, 300

Software Support Plan .............................................. 306

Spiral Objectives ....................................................... 15

Staff Preparation ....................................................... 300

Staffing .......................................................... 204, 206, 237, 272, 314

Stakeholder Concurrence ........................................... 232

Static–Structure Diagrams 18, 21, 74, 90, 94, 96, 100, 119, 136, 147, 148, 170, 171, 190, 192

Status Reports .................................................. 207, 243, 248, 327

Support Objectives ................................................... 307

Support Resources ................................................... 310

Support Responsibilities ........................................... 309

Support Set .......................................................... 244

System and Software Architecture Description.. 1, 66, 201, 209, 210, 226, 240, 241, 243, 270, 312, 325, 328

System and Software Requirements Definition.. 1, 49, 52, 209, 210, 220, 240, 243, 248, 270, 312, 314, 324, 325, 328

System Boundary ..................................................... 8, 13, 31

System Design ....................................................... 67

System Priorities ..................................................... 197, 200, 233

System Requirements 3, 31, 51, 56, 57, 58, 59, 62, 68, 196, 228, 229, 233, 273, 304, 314

System Shortfalls ................................................... 238
<table>
<thead>
<tr>
<th><strong>Guidelines for MBASE</strong></th>
<th><strong>Index</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology Evolution</td>
<td>Transition Plan 9, 50, 55, 56, 202, 209, 210, 211, 218, 244, 298, 306, 308</td>
</tr>
<tr>
<td>Test Cases</td>
<td>Transition Strategy</td>
</tr>
<tr>
<td>Test Incident Reports</td>
<td><strong>UML Component</strong></td>
</tr>
<tr>
<td>Test Plan</td>
<td>Unit Testing</td>
</tr>
<tr>
<td></td>
<td>Upcoming Changes</td>
</tr>
<tr>
<td>Test Procedure</td>
<td>Use-Case</td>
</tr>
<tr>
<td>Test Summary</td>
<td>User interface</td>
</tr>
<tr>
<td>Tool Support</td>
<td>User Manual</td>
</tr>
<tr>
<td>Training</td>
<td>Verification and Validation</td>
</tr>
<tr>
<td></td>
<td>Workload Evolution</td>
</tr>
<tr>
<td>Transition Cost</td>
<td></td>
</tr>
</tbody>
</table>