POSSIBLE UML-BASED SIZE MEASURES

Richard D. Stutzke

Science Applications International Corporation
6725 Odyssey Drive
Huntsville, AL 35806-3301
(256) 971-6624 (office)
(256) 971-6550 (facsimile)

Richard.D.Stutzke@cpmx.saic.com

07 October 1998

Presented at the Thirteenth International Forum on COCOMO and Software Cost Modeling, Los Angeles.

© 1998 by Richard D. Stutzke
TOPICS

- Motivation
- UML Notation
- Objectives
- Historical Approaches
- Proposed Approach
MOTIVATION

- We build software products in new ways
  - Package components as “objects”
  - Assemble products from existing components (reuse)
  - Use intelligent tools reduce or even eliminate coding

⇒ Object-oriented analysis and design, plus testing, consume most of the effort

- We need ways to estimate effort from high level design elements
  - Logical design (problem requirements)
  - Physical design (implementation requirements)
  - Production process (knowledge, skill, tools, reuse)
UNIFIED MODELING LANGUAGE (UML)

- **Design Goals**
  - Expressive visual model language
  - Integrate best practices
  - Independent of programming language
  - Extensible and customizable

- **Uses**
  - Specification
  - Visualization
  - Documentation
  - Construction and generation of software
UML NOTATION*

- Diagram elements
- Static Structure Diagrams
- Use Case Diagrams
- Sequence Diagrams
- Collaboration Diagrams
- Statechart Diagrams
- Activity Diagrams
- Implementation Diagrams
- Model Management and Extension Mechanisms

*From “UML Notation Guide”, version 1.1, 1 September 1997
DEFINING A SIZE MEASURE

• Goals
  - Correlated to development effort
  - Independent of technology
  - Can be estimated early
  - Can be estimated easily

• Approach
  - Focus on logical design
  - Hide implementation in the productivity
  - Use UML elements produced during analysis
  - Use a subset of the UML elements
CHOOSING ORTHOGONAL COMPONENTS

• Three Facets of Software [DeMarco, 1982]
  - Data (content and structure)
  - Function (algorithms)
  - Behavior (control)

• Object Facets
  - Attributes (data)
  - Behavior (operations or methods)
  - State (memory)
  - Identity (a key – a data attribute)
  - Responsibilities (“purpose”, raison d’être)

Issue: relative contribution of each component
(i.e., how combine to get a single value)
SEQUENCE OF GENERATING MODEL COMPONENTS
(SHLAER-MELLOR METHOD)

1. Object (class) Diagram and Data Dictionary
2. Event Dictionary and Event Scenarios
3. State Diagram (for each object)
4. Data Flow diagram (for each state)
5. Event trace Diagram (links events and objects for each scenario)
6. Class Interaction Diagram
7. Context Diagram

Items 5-6 are started early and revised. They help organize the analysis process.
OOA MODEL COMPONENTS

Object

Object Diagram

Data Dictionary
Vehicle ID = integer, range 1-10
Velocity = speed in mph, range 0-100
Answer = [T | F]

Dynamic

Event Analysis
Key Event Dictionary
Event Scenario

State Diagram

Event Trace Diagram

Functional

Class Interaction DFD

Context Diagram
UML COMPONENTS FOR DATA

• Class diagram shows:
  - Class
  - Class attributes
  - Relations/associations (multiplicity and conditionality [obligation] only. Ignores fidelity.)
  - Aggregation and composition
  - Generalization and specialization

• Collaboration diagram shows:
  - Collaboration (communication)
  - Message syntax

• Sequence diagram shows:
  - Flow of messages between objects in time sequence (event trace diagrams, scenarios)

• Use case diagram shows:
  - Collaboration between system and actors in the environment
  - Relations between use cases
UML COMPONENTS FOR FUNCTION

- **Sequence diagram shows:**
  - Flow of messages between objects in time sequence
    (event trace diagrams, scenarios)

- **Use case diagram shows:**
  - Collaboration between system and actors in the environment
  - Relations between use cases

- **Collaboration diagram shows:**
  - Collaboration (communication)
  - Message syntax
UML COMPONENTS FOR CONTROL

- **Statechart shows:**
  - Sequence of states for a reactive class
  - Stimuli and response(s) thereto
  - Includes nested, sequential, and orthogonal states (orthogonal = independent, nonconcurrent)

- **Activity diagram shows:**
  - Transitions taken when a state activity is completed
  - A special case of statecharts
  - Good for forks and joins (rendezvous, task synchronization)

- **Implementation diagrams show:**
  - Run-time dependencies of components (e.g., rendezvous, task synchronization)
  - Packaging structure of components of the deployed system (allocation of components to nodes, physical topology)
## HISTORICAL PERSPECTIVE

<table>
<thead>
<tr>
<th>Method</th>
<th>Data</th>
<th>Function</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Bang [DeMarco, 1982]</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Data Bang [DeMarco, 1982]</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature Points [Jones, 1988]</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Mark II Function Points [Symons, 1988], [Douglas, 1995]</td>
<td>x</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>ASSET Function Points [Reifer, 1990]</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>MOOSE [Chidamber, 1994]</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Object Points [Boehm, 1995]</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3D Function Points [Whitmire, 1995]</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>FPA and Use Cases [Armour, 1996]</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPA [Gupta, 1996]</td>
<td>x</td>
<td>x</td>
<td>?</td>
</tr>
<tr>
<td>Predictive Object Points [Minkiewicz, 1997]</td>
<td>x</td>
<td>x</td>
<td>?</td>
</tr>
</tbody>
</table>
DATA SIZE FROM CLASS DIAGRAM [DeMARCO, 1982]

\[
\text{Sizedata} = \sum_{i=1}^{\text{#Classes}} W(R_i) \quad \text{where } R_i = \# \text{ relations for class } i
\]

and the weights are defined by:

<table>
<thead>
<tr>
<th>( R_i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>W(( R_i ))</td>
<td>1.0</td>
<td>2.3</td>
<td>4.0</td>
<td>5.8</td>
<td>7.8</td>
<td>9.8</td>
</tr>
</tbody>
</table>

But this model ignores the internal data content (the attributes).
DATA SIZE FROM CLASS DIAGRAM [DOUGLAS, 1995]

- **Input Parameters**
  - $N_e = \text{Number of entities (classes)}$
  - $N_a = \text{Number of attributes (average per entity)}$
  - $N_r = \text{Number of relations (total)}$

- **Derived Parameters (based on several assumptions)**
  - $T_{if} = \text{Total input fields}$
    - $= 2*N_a*(1 + N_r/N_e) + 2*N_e$
  - $T_{ea} = \text{Total entities accessed}$
    - $= 4*N_e + 8*N_r$
  - $T_{of} = \text{Total output fields}$
    - $= N_a*(1 + N_a/N_e) + 3*N_e$

which gives:

$$UFP = 0.58*T_{if} + 1.66*T_{ea} + 0.26*T_{of}$$
DATA SIZE FROM CLASS DIAGRAM [DOUGLAS, 1995] (CONTINUED)

- Expanding gives:

\[\text{UFP} = 1.42\left[ N_A (1 + \frac{N_R}{N_E}) + 6.04 N_E + 9.35 N_R \right] \]

\(\%\) Average “hairiness” of the object

\[\approx 1.42\left[ N_a + 6 N_e + 9 N_r + N_a N_r/N_e \right] \]

- Observations based on this model:
  - Attributes are “cheap” (maybe DeMarco was right to ignore them?)
  - Relations “cost” 50 % more than entities
  - A measure of entity complexity might be: \(N_a N_r/N_e\)
RELATIONS AFFECT DATA AND FUNCTIONS

- Characteristics of Relations
  - Multiplicity ("How many instances are connected?")
  - Conditionality ("obligation")
  - Fidelity ("How many such relations can exist at once?")

- The static structure ("data") is characterized by $N_e$, $N_a$, $N_r$.

- The dynamics ("functions") enforces the policies:
  "Cost" $\propto$ Number of policies enforced?
ACTIVE OBJECTS

• Definition
  - Autonomously performs actions
  - Coordinates the activities of component objects
  - Generates events

• Examples
  - Produce or analyze data ("functions")
  - Produce or control actions ("dynamic behavior")

• Consequences
  - Have a State Model (→ Statechart)
  - Have an action specification (state diagrams [contain p-specs], scenario diagrams, notes)
STATE SIZE: ACTIVE CLASSES ONLY*

- Option # 1: Count number of events in event dictionary, $N_{ed}$:
  
  \[ \text{Size}_{state} = 2*N_{ed} \]  
  (assumes one event per state)

- Option # 2: Count similarly to data (class diagram):
  
  \[ N_s = \text{Number of states ("bubbles")} \]
  \[ N_t = \text{Number of transitions ("lines" connecting bubbles)} \]

Assuming each takes equal effort to define gives:

\[ \text{Size}_{state} = N_s + N_t \]

We probably need to adjust the weights upward by 50% or so (based on [Whitmire, 1995]).

*Passive objects have no behavior.
FUNCTION SIZE

- “Relation Management” already included in data size

- Most operations are simple
  - Described in statechart (similar to a process specification)
  - Included in state size

- Complicated operations counted separately
  - Use COCOMO’s “complexity” criteria to identify them. (Compute a weighted sum. Ignore if total below a threshold.)
  - Set size to, say, 3 feature points. (Possibly weight as a function of complexity.)
SUMMARY: ELEMENTS TO USE

- Class diagram (OIM, ERD)
  - Number of entities
  - Number of attributes
  - Number of relations
  - Adjust for entity “complexity” \((N_a \cdot N_r / N_e)\)
  - Adjust for “relation management” (policies enforced)

- State Charts
  - Number of states
  - Number of transitions
  - No apparent measure of “behavioral complexity”

- Analysis Notes
  - Identifies “complicated operations” (3 Feature Points each)
  - COCOMO’s complexity ratings perhaps usable
COMBINING THE SIZES

\[ \text{Size}_{\text{total}} = W_d \cdot \text{Size}_d + W_s \cdot \text{Size}_s + W_f \cdot S_f \]

where \( W_d, W_s, \) and \( W_f \) are weights to be determined.

More complex functional forms are not justified given the lack of empirical data.
EFFORT ALLOCATION*

- Breakdown of different OOA/RD activities:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition domains, build Domain chart</td>
<td>2%</td>
</tr>
<tr>
<td>Build, review Object Information Models</td>
<td>21%</td>
</tr>
<tr>
<td>Build, review State Models</td>
<td>10%</td>
</tr>
<tr>
<td>Build, review Action Specification</td>
<td>12%</td>
</tr>
<tr>
<td>Develop and test Software Architecture and Translation rules</td>
<td>10%</td>
</tr>
<tr>
<td>Manually translate OOA models into code, integrate and test</td>
<td>25%</td>
</tr>
</tbody>
</table>

- Total OOA time: 65%

- Breakdown of the OOA modeling effort:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development</td>
<td>77%</td>
</tr>
<tr>
<td>Review</td>
<td>12%</td>
</tr>
<tr>
<td>Updating</td>
<td>11%</td>
</tr>
</tbody>
</table>

- Administrative activities for the project consumed an additional 14%

*From Appendix A of [Montrose, 1996]. Data obtained on an embedded product written in C++ with over 200 classes using the Shlaer-Mellor OOA/RD method.
MANY QUESTIONS REMAIN....

- How much effort is needed to reuse portions of object models?
- How does the particular analysis method affect effort?
- How much effort is needed to prepare the “implementation-specific” diagrams?
- How much effort is needed to map the design objects into working code?
- How does test effort depend on UML size?
## GLOSSARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Diagram</td>
<td>Represents the classes and the associations that govern class behavior, and their associated information.</td>
</tr>
<tr>
<td>Data Dictionary</td>
<td>A set of class, attribute, and association descriptions that support the Object Model.</td>
</tr>
<tr>
<td>Event</td>
<td>A message passed to a class as the result of a real-world occurrence.</td>
</tr>
<tr>
<td>Key Event Dictionary</td>
<td>A table outlining system responses to environmental events, including affected classes.</td>
</tr>
<tr>
<td>Event Scenario</td>
<td>An interesting, time-ordered list of events from the system environment.</td>
</tr>
<tr>
<td>State Diagram</td>
<td>Represents the lifecycle of a class and the events that drive its behavior.</td>
</tr>
<tr>
<td>Data Flow Diagram (DFD)</td>
<td>Represents the functional processing required when entering a class state.</td>
</tr>
<tr>
<td>Event Trace Diagram</td>
<td>A summary of the time-ordered, asynchronous event communication between classes.</td>
</tr>
</tbody>
</table>
GLOSSARY (CONTINUED)

Class Interaction DFD) - Represents a summary of both synchronous data access and asynchronous events sent between classes.

Context Diagram - Represents system interfaces with its environment.