Estimating Software Size

developed and presented by

Richard E. (Dick) Fairley
Professor and Director
of Software Engineering
Oregon Graduate Institute
dfairley@cse.ogi.edu
Biographical Statement
Richard E. Fairley, Ph.D.

Dr. Richard E. (Dick) Fairley is a professor and Director of Software Engineering at the Oregon Graduate Institute in Beaverton, Oregon. Dr. Fairley is also involved in design and implementation of the Oregon Master of Software Engineering program - a state-funded collaborative program among four Oregon universities. He is founder and principal associate of Software Engineering Management Associates, Inc. (SEMA), a firm specializing in consulting services and training in software systems engineering, software project management, software cost estimation, project planning and control techniques, software risk management, and software process assessment and improvement. Prior to joining OGI, Dr. Fairley was Dean of Computer Science at Colorado Technical University in Colorado Springs, Colorado. Dr. Fairley has more than 20 years experience as a university professor, researcher, consultant, and trainer in software engineering. He has designed and implemented educational programs in universities and in industry, lectured and consulted with many companies world-wide, and headed research programs in software engineering.

In 1988-89, Dr. Fairley held a two year appointment as a Distinguished Visiting Scientist at the Jet Propulsion Laboratory and consultant to the NASA Headquarters software quality group. He is past chairman of the advisory committee to the Education Division of the Software Engineering Institute. He is a past member of program committee for the annual SEI Risk Conferences, and a former Distinguished Visiting Professor at Drexel University. From 1984 through 1993 he was program chair for the COCOMO Cost Estimation Group's annual meetings.

Dr. Fairley is author of the text book *Software Engineering Concepts*, editor of three texts, author/presenter of several videotape series in software engineering, and a principal author of ANSI/IEEE Standard 1058 for Software Project Management Plans and ANSI/IEEE Standard 1362 for Operational Concept Documents. He is the author of numerous research papers on a variety of topics in computer science and software engineering. Dr. Fairley has Bachelors and Masters degrees in Electrical Engineering. His Ph.D. in computer science is from UCLA. Prior to obtaining his Ph.D., Dr. Fairley worked in industry as an electrical engineer and computer programmer.
Outline

• Session 1: Size Measures
• Session 2: Size Estimation Techniques
ESTIMATING SOFTWARE SIZE

SESSION 1: SIZE MEASURES

developed and presented
by
Dr. Richard E. Fairley
Professor and Director of Software Engineering
Oregon Graduate Institute
dfairley@cse.ogi.edu

SESSION OVERVIEW

- Some Fundamentals
- Some Size Measures
  - Lines of Code
  - Pages of Documentation
  - Number of Requirements
  - Size of Design
  - Function Points
  - Domain-Specific Measures
  - Object Points
WHY MEASURE (anything)?

- Two fundamental reasons:
  1. to assess current status
     - by comparing to planned status
  2. to provide a basis for predicting the future
     - of the current project
     - of a future project

THE FUNDAMENTAL AXIOM OF ESTIMATION

- All estimates are projections from past to future, suitable adjusted to account for differences between past and future
  - the past is captured by historical data relating product attributes to effort, schedule, and resource requirements
  - the future is summarized by the attributes of the product to be built or modified
  - the differences are adjustment factors to account for:
    - different customer
    - different workers
    - different tools
    - different application
    - and so forth
ELEMENTS OF ESTIMATION

Future-Product Attributes

Adjustment Factors

Assumptions and Constraints (risk factors)

Historical Data

Estimate

AN EXAMPLE

- Suppose we prepare an estimate using simple analogy and rule of thumb:
- By analogy, we estimate the size of the future product will be 60,000 lines of code
- Using our productivity rule of thumb (500 LOC/SM), we estimate the project will require
  
  \[
  \frac{60,000}{500} = 300 \text{ Staff-Months of Effort}
  \]
  - we might use 15 people to complete the project in 20 months, assuming similar staffing in the past
- NOTES:
  1. the rule-of-thumb summarizes our history
  2. the analogy provides the future-product attributes
  3. no adjustment factors are applied
A NOTE ON DERIVING CONVERSION FACTORS

- Productivity factors such as the previously cited 500 LOC/SM and other conversion factors of interest can be derived by examining local, relevant historical data.
- Such factors provide the bridge from our "surrogate" measures, such as lines of code, to factors of interest such as estimated effort, schedule, and defect density.
- These conversion factors are useful in the aggregate but should never be applied to individual staff members or individual modules of code.

ANOTHER EXAMPLE

- Using COCOMO 1.0:
  - the regression equations relating size, effort, and schedule capture our history
  - the size estimate summarizes the future product
  - the cost drivers allow adjustments for differences between past and future

Copyright © 1998 by Richard E. Fairley

Size-Estim chart 1-7
WHY MEASURE PRODUCT ATTRIBUTES?
(why not rely on analogy and expert judgment?)

- We cannot build an estimation model based on historical data without some description of past-product attributes
- We cannot make tradeoffs among schedule, resources, quality, and product attributes without some measures of product attributes
- We cannot measure productivity, quality, or progress without some measure of output produced

Measurement of product attributes is a "means to an end"

BUILDING A HISTORICAL DATABASE

- Historical data is required to make accurate estimates
- Consistent counting rules across projects (and organizations) are necessary to build a meaningful history

The more local, consistent, and relevant the historical data, the better the estimates
THE IMPORTANCE OF CONSISTENT COUNTING RULES

- Some Lines-of-Code (LOC) examples:
  - Executable lines only: 1000
  - Executable plus declarations: 1200
  - Executable, declarations, and comments: 1500
  - Plus test harnesses and drivers: 3000
  - plus library routines and reused code: ?
  - Multiple statements per line = one LOC?
  - Multiple statements per line = multiple LOCs?

Without consistent counting rules, different people will count the size of a program differently

WHAT PRODUCT ATTRIBUTES ARE OF INTEREST?

- Product attributes that influence estimates (and actuals) include:
  - software size
  - software complexity
  - product familiarity
  - quality attributes
  - design constraints
    - interfaces
    - performance requirements
    - memory usage
    - hardware platform
WHY FOCUS ON SOFTWARE SIZE?

1. Software size correlates more strongly with the "means to an end" factors of interest than any other product attribute:
   - Effort and Schedule: larger systems require more time and effort
   - Quality: larger systems present more opportunities to make mistakes
   - Maintainability: larger systems are harder to understand, modify, and test

2. Software size can be measured more objectively than other product attributes

THE INFLUENCE OF PRODUCT COMPLEXITY

Increasing Effort & Schedule
THE INFLUENCE OF COMPLEXITY - II

- Complexity is dealt with by focusing on individual domains and making relative adjustments within those domains; for example:
  - the COCOMO 1.0 “modes” and the CPLX factor
  - Function Points and Environmental Factors
  - other “Function Point like” measures and their adjustment factors

MEASURING SOFTWARE COMPLEXITY

- Measuring the impact of software complexity on estimated effort, schedule, and quality is somewhat subjective
  - it depends on our familiarity with the application domain
- Complexity measures include:
  - Boehm's complexity adjustment factor
    - effort and schedule adjustment
  - Constantine’s coupling and cohesion
    - design complexity
  - McCabe’s cyclomatic complexity
    - control flow complexity
Boehm* provides a four-element measure of product complexity:
- complexity of the control operations
- complexity of the computations
- complexity of data management
- complexity of the device-dependent operations

A table is used to select a complexity adjustment factor and increase or decrease the estimated effort and schedule for a project

\[ 0.7 \leq CPLX \leq 1.65 \]


SOME SIZE MEASURES

- Lines of Code
- Pages of Documentation
- Number of Requirements
- Size of Design
- Function Points
- Domain-Specific Measures
- Object Points
SOME SIZE ESTIMATION TECHNIQUES

- Size estimation techniques include:
  - Analogy
  - Rule of Thumb
  - Expert Judgment
  - The Delphi Approach
  - Constrained Estimation
  - Algorithmic Procedures
  - Statistical Techniques

Any of these techniques can be applied to any of the size measures
- SEE SESSION 2 -

SOME SIZE MEASURES

=> Lines of Code
  - Pages of Documentation
  - Number of Requirements
  - Size of Design
  - Function Points
  - Domain-Specific Measures
  - Object Points
PROBLEMS WITH LINES OF CODE

- It is difficult to accurately estimate lines of code
  - growth of 30% or 40% over the original estimate is not uncommon
- Uniform counting rules must be used
  - is a line of code an end-of-line? a semicolon? a comment?
- Uniform counting conventions must be used
  - executables only: 1000 LOC
  - plus declarations: 1200 LOC
  - plus comments: 1500 LOC
  - plus test stands: 3000 LOC
  - plus library code: ???
  - plus reused code: ???

Copyright © 1998 Size-Estim by Richard E. Fairley chart 1-21

TRACKING ESTIMATED LINES OF CODE

Estimated source lines of code (KSLOC)

<table>
<thead>
<tr>
<th>1st Qtr</th>
<th>2nd Qtr</th>
<th>3rd Qtr</th>
<th>4th Qtr</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>new</td>
<td>modified</td>
<td>reused</td>
<td></td>
<td>total</td>
</tr>
</tbody>
</table>

Copyright © 1998 Size-Estim by Richard E. Fairley chart 1-22
SOME SIZE MEASURES

- Lines of Code
  => Pages of Documentation
- Number of Requirements
- Size of Design
- Function Points
- Domain-Specific Measures
- Object Points

MEASURING SIZE BY PAGES OF DOCUMENTATION

Pages of documentation can be used to estimate some aspects of project effort

- For example:
  - 10 pages of requirements specification
  - 50 pages of design documentation
  - 20 pages of test plans and procedures
  - 50 pages of user manual
  - 30 pages of operations and maintenance procedures

A conversion factor of staff-hours per page or dollars per page might be used for various types of documents
SOME SIZE MEASURES

- Lines of Code
- Pages of Documentation
=> Number of Requirements
- Size of Design
- Function Points
- Domain-Specific Measures
- Object Points

COUNTING THE NUMBER OF REQUIREMENTS

Some techniques:
- count the number of pages in the requirements document
- count the number of "shall"s in the requirements document
- count the number of weighted "shall"s in the requirements document
- count the number of unique tasks the system must perform
- count the number of use cases
- count the number of operational scenarios

normalization to some standard or convention is required in all cases

Copyright © 1998 by Richard E. Fairley

Size-Estim chart 1-25

1-13
PROBLEMS WITH COUNTING REQUIREMENTS

- Requirements must be decomposed until:
  - all hidden complexities are exposed
  - opportunities for reuse are identified
  - weighting factors for implementation difficulty can be assigned to the requirements
- The impact of non-functional requirements is difficult to estimate:
  - performance constraints
  - interfaces
  - quality requirements
    - safety, security, reliability, ease of use

WAYS TO COUNT REQUIREMENTS

- Functional Techniques
  - lowest-level bubbles in the data-flow diagram
  - number of elements in the data dictionary
- Data-Oriented Techniques
  - entities and relations in the entity-relationship diagrams
- State-Oriented Techniques
  - states and transitions
- Object-Oriented Techniques
  - number of objects, attributes, and interactions
DERIVED SIZE MEASURES

- We can "size" a system by deriving design and code size from weighted requirements
- For example:
  - 300 equally-weighted requirements might result in:
    - 30,000 lines of code @ 100 LOC per requirement
    - 1000 modules @ 30 LOC per module
  - 1000 modules might require 1000 staff-weeks @ 1 staff-week per module
  - 20 people might be required for 50 weeks
  - $3M would be required @ $3,000 per staff-week (loaded salaries)

SOME SIZE MEASURES

- Lines of Code
- Pages of Documentation
- Number of Requirements
  => Size of Design
- Function Points
- Domain-Specific Measures
- Object Points
SIZE OF DESIGN

- For functional (top-down) designs, we can count:
  - the number of modules
  - the number of interfaces
  - the depth of the calling tree
  - fan-in and fan-out at each level

DeMarco's BANG Metrics

- Function Bang
  - count number of lowest-level bubbles in the data-flow diagram
  - weight according to type of functional primitive and number of data items used by the function
  - can be converted to lines of code; for example:
    \[ \text{LOC} = 300 + 50 \times \text{FBC} \] (Functional Bang Count)

- Data Bang
  - count the number of entities in the Entity-Relation Diagram
Attributes of Object-Oriented Systems include:
- number of objects
- number of methods
- interactions among objects
- depth of the inheritance tree
- aggregation structures

Chidamber and Kemerer's OO Metrics*
- Weighted methods per class (WMC)
  - weighted by complexity of methods in a class
- Depth of inheritance tree (DIT)
  - ancestor classes that can influence a class
- Number of children per class (NOC)
  - number of immediate successors
- Coupling between object classes (CBO)
  - number of classes coupled to a class
- Response per class (RFC)
  - local methods plus number of methods invoked
- Lack of cohesion metric (LCOM)
  - relation of local methods to local variables

*IEEE Trans. on Software Engineering (20) 5, 1994
DESIGN COMPLEXITY

Computing design complexity involves:
1. Computing the cyclomatic complexity of each module
2. Computing the structural complexity of module interfaces
2. Computing the composite complexity of the system

1. CYCLOMATIC COMPLEXITY

- The structural complexity of a software module is determined by computing the cyclomatic complexity of each module’s control-flow graph
- The cyclomatic complexity, C, of a module, M, is computed as:

  \[ C(M) = E - N + 2 \]

  where \( N \) is the number of nodes in the control-flow graph and \( E \) is the number of connections between nodes
- For example:

  \[ C(M) = 12 - 10 + 2 = 4 \]
2. INTERFACE COMPLEXITY

The structural complexity of a collection of software modules is called the Design Complexity.

Design Complexity is computed in two steps:

1. Compute the Design Complexity of each module, by considering only the nodes that have interfaces to other modules:

   \[ DC(M) = 7 - 6 + 2 = 3 \]

2. Compute the Design Complexity of the collection of modules, \( S \), as:

   \[ DC(S) = \sum DC(M_j) - N + 1 \]

   where \( DC(M_j) \) is the design complexity of module \( j \) and \( N \) is the number of modules.

   \[ DC(S) = 6 - 4 + 1 = 3 \]
SOME SIZE MEASURES

- Lines of Code
- Pages of Documentation
- Number of Requirements
- Size of Design
=> Function Points
- Domain-Specific Measures
- Object Points

FUNCTION POINT SIZING

- Function point sizing was developed by Alan Albrecht and colleagues at IBM in 1979
- FP sizing is based on counting five types of product attributes:
  - number of unique inputs
  - number of unique outputs
  - number of files
  - number of unique database interactions
  - number of interfaces to other software
FUNCTION POINT WEIGHTING

- A simple, average, or complex weighting factor is applied to each function point

<table>
<thead>
<tr>
<th></th>
<th>Simple</th>
<th>Average</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs:</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Outputs</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Queries</td>
<td>3</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Files</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Interfaces</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>
A FUNCTION POINT ESTIMATION PROCEDURE

1. Count the number of function points
2. Apply the weighting factors
3. Sum up the number of function points
4. Specify appropriate values for 14 "environmental factors"
5. Compute the complexity adjustment factor
6. Compute the number of adjusted function points
7. Convert to lines of code (optional)
8. Apply a rule-of-thumb productivity factor to determine required effort
9. Convert effort into people and time

A FUNCTION POINT EXAMPLE

<table>
<thead>
<tr>
<th>Function Point Weighting Table</th>
<th>simple</th>
<th>average</th>
<th>cmplex</th>
<th>FPs</th>
</tr>
</thead>
<tbody>
<tr>
<td># inputs</td>
<td>3 x 3</td>
<td>2 x 4</td>
<td>0 x 6</td>
<td>17</td>
</tr>
<tr>
<td># outputs</td>
<td>3 x 4</td>
<td>6 x 5</td>
<td>3 x 7</td>
<td>63</td>
</tr>
<tr>
<td># inquiries</td>
<td>0 x 3</td>
<td>20 x 5</td>
<td>3 x 7</td>
<td>121</td>
</tr>
<tr>
<td># files</td>
<td>5 x 7</td>
<td>2 x 10</td>
<td>0 x 15</td>
<td>55</td>
</tr>
<tr>
<td># interfaces</td>
<td>0 x 5</td>
<td>0 x 7</td>
<td>3 x 10</td>
<td>30</td>
</tr>
</tbody>
</table>

Total FP: 286
THE FP ENVIRONMENTAL FACTORS

- Communication links
- Distributed processing
- Performance objectives
- Computer capacity
- Transaction rate
- On-line queries
- End user efficiency
- On-line updates
- Complex processing
- Code reusability
- Installation ease
- Operational ease
- Multiple sites
- Ease of change

The Environmental Factors

- Values in the range of 0 to 5 are chosen for each of the 14 factors
- A value of 0 indicates much less influence than in the typical project
- A value of 5 indicates much more influence than in the typical project
- A value of 2.5 indicates similar influence as on past projects
SPECIFY VALUES FOR THE ENVIRONMENTAL FACTORS

- Data Communications 2
- Distributed Processing 4
- Performance Objectives 4
- Heavy Use 4
- Transaction Rate 4
- Reusability 5
- Operational Ease 5
- Multi-Site Use 5
- Ease of Change 5
- All Others \( \frac{3}{3} \) each

TOTAL: \( N = 53 \)

COMPUTE THE COMPLEXITY ADJUSTMENT FACTOR

- Complexity Adjustment Factor (CAF):
  \[ CAF = (0.65 + 0.01 \times N) \]
  \[ = (0.65 + 0.01 \times 53) = 1.18 \]

- Adjusted Function Points (AFP)
  \[ AFP = (CAF \times FP) = 1.18 \times 286 = 337 \]

The purpose of counting rules, weighting factors and complexity adjustment factors is to normalize all adjusted function points into the same “size” range.
APPLY A PRODUCTIVITY FACTOR TO OBTAIN ESTIMATED EFFORT AND SCHEDULE

- Assume the productivity rule-of-thumb for our organization is 3.5 FP/SM
- 337 / 3.5 = 96 staff-months of effort
- The project might be scheduled as 8 people for 12 months
  - or 12 people for 9 months
  - but not 96 people for one month

3.5 FP/SM is the productivity level to be measured in the coming project

CONVERTING FUNCTION POINTS TO LINES OF CODE (optional)

#Lines-of-Code = #Function Points * Conversion Factor

The conversion factor is language dependent:

<table>
<thead>
<tr>
<th>Language</th>
<th>Conversion Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler</td>
<td>320 LOC/FP</td>
</tr>
<tr>
<td>C</td>
<td>128 LOC/FP</td>
</tr>
<tr>
<td>FORTRAN77</td>
<td>105 LOC/FP</td>
</tr>
<tr>
<td>COBOL</td>
<td>105 LOC/FP</td>
</tr>
<tr>
<td>PL/1</td>
<td>80 LOC/FP</td>
</tr>
<tr>
<td>Ada</td>
<td>70 LOC/FP</td>
</tr>
<tr>
<td>Lisp / Prolog</td>
<td>64 LOC/FP</td>
</tr>
<tr>
<td>4 GLs</td>
<td>20 LOC/FP</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>6 LOC/FP</td>
</tr>
</tbody>
</table>

NOTE: the conversion factor should be locally derived
NOTES

1. Conversion factors from the size measure (function points or others) to lines of code should be locally derived.

2. Size measures are converted to lines of code using local historical data because lines of code can be measured unambiguously in existing software by some standardized counting rules.

CONVERTING FUNCTION POINTS TO LINES OF CODE - II

• Adjusted Function Points = 337
  
  COBOL LOC = 337 * 105 = 35,385 LOC
  
  4GL LOC = 337 * 20 = 6,740 LOC
VARIATIONS ON FUNCTION POINTS

• Mark II Function Points (Charles Symons)
  - tailored to "logical transactions"
• Feature Points (Capers Jones)
  - tailored to computational systems

Copyright © 1998 by Richard E. Fairley
Size-Estim chart 1-53

OTHER FUNCTION POINT "LIKE" MEASURES

• Transaction Processing Systems
  – number of transactions (Mark II Function Points)
• Computational Systems
  – number of inputs, outputs, files, interfaces, and algorithms (feature points)
• User Interfaces
  – number of screens, windows, menus
• Process Control
  – number of valves, actuators, temperature sensors, pressure sensors
• Object-Oriented Systems
  – number of objects, methods, relations; depth of the inheritance tree; aggregation

Copyright © 1998 by Richard E. Fairley
Size-Estim chart 1-64

1-27
SOME SIZE MEASURES

- Lines of Code
- Pages of Documentation
- Number of Requirements
- Size of Design
- Function Points

=> Domain-Specific Measures
- Object Points

DOMAIN_SPECIFIC APPROACHES TO SOFTWARE ESTIMATION

- User_Interfaces:
  Effort = f(#screens, #menus, #fields)
- Process_Control:
  Effort = f(#sensors, #actuators)
- Databases:
  Effort = f(#entities, #attributes, #relations)
- OO_Systems:
  Effort = f(#object, #methods, #protocols)
THE VERNER-TATE SIZING EQUATIONS*

- For menus:
  \[ \text{LOC} = 68.1 + 3.3 \times \text{# of items} \]
- For inputs:
  \[ \text{LOC} = 105 + 23.4 \times \text{# of data elements} + 9.7 \times \text{# of files} \]
- For reports:
  \[ \text{LOC} = 75.8 + 6.7 \times \text{# of data elements} + 58.5 \times \text{# of files} \]


COUNTING DOMAIN-SPECIFIC FACTORS

- Ground-based mission support software might be categorized by sub-domains:
  - mission planning
  - command generation
  - spacecraft performance monitoring
  - telemetry processing
  - database management
  - payload data processing and distribution
  - ground station control and monitoring
THE CHALLENGE

- What "external" sizing factors can you count in your domain?
- The external sizing factors should:
  1. be countable at the requirements / early design level
  2. characterize the software to be implemented
  3. be convertible to factors of interest - based on available, local history; for example
     - effort
     - schedule
     - defects
     - performance

THE APPROACH

1. Identify the "external sizing factors"
2. Develop standard counting rules
3. Analyze past systems to develop conversion factors from "external size" to lines of code, effort, schedule, quality, etc
4. Count factors of "external size"
5. Apply the conversion factors into lines of code and from lines of code to factors of interest
6. Develop adjustment factors to account for variations in effort, schedule, quality for historical systems of similar "essential size"
   - to explain why different systems of similar "external size" exhibited different effort, schedule, quality, and so forth
ELEMENTS OF ESTIMATION

Future-Product Attributes

Adjustment Factors

Assumptions and Constraints (risk factors)

Historical Data

Estimate

Estimation Procedure

SOME SIZE MEASURES

- Lines of Code
- Pages of Documentation
- Number of Requirements
- Size of Design
- Function Points
- Domain-Specific Measures
  => Object Points
COCOMO 2.0 OBJECT POINTS

- COCOMO 2.0 has three models containing increasing levels of detail
  - Stage 1: Application Composition and Early Prototyping Stage
  - Stage 2: Early Design Stage
  - Stage 3: Post-Architecture Stage

COCOMO 2.0 Stage Models

- Stage 1: Application Composition and Early Prototyping stage
  - size estimate based on Object Points
  - no cost drivers
- Stage 2: Early Design stage
  - size estimate based on Unadjusted Function Points
  - seven cost drivers
- Stage 3: Post-Architecture stage
  - size estimate based on Function Points, Lines of Code, or other measure
  - seventeen cost drivers
STAGE1 MODEL: APPLICATION COMPOSITION AND PROTOTYPING

- Application Composition
  - systems that can be built from interoperable components
- Examples include:
  - GUI builders
  - database managers
  - transaction processing middleware
  - hypermedia handlers
  - application generators
  - domain-specific components
    - e.g., financial, medical, industrial process control domains

OBJECT-POINT COUNTING

- "Object Points" define screens, reports, and 3GL modules as objects (similar to counting function points or Verner-Tate size metrics)
  - object points are similar to what was characterized earlier as "external size"
  - "The term "object" may or may not have any relationship to the use of the term in "object oriented"
APPLICATION COMPOSITION PROCEDURE

Estimation at the Application Composition level is a seven step process:
1. Estimate the number of Object Points in the system
2. Classify Object Points into simple, medium, and complex categories
3. Apply weighting factors
4. Sum up the total number of Object Points (OPS)
5. Estimate the number of New Object Points (NOPS), taking into account the percent of reuse for the project
6. Apply a productivity factor
7. Compute the estimated person-months

DETERMINING THE NEW OBJECT POINTS (NOPS)

NOPS = \[(OPS * (100 - \%REUSE)) / 100\]

where \%REUSE is computed using the COCOMO 2.0 reuse model

note: NOPS = OPS when \%REUSE = 0
NOPS = 0 when \%REUSE = 100

In COCOMO 2.0 “reuse” includes adaptation of existing code
THE COCOMO 2.0 REUSE MODEL

• The factors in the reuse model are:
  – the Software Understanding increment (SU)
  – the Assessment and Assimilation factor (AA)
  – the percent of design to be modified (DM)
  – the percent of code to be modified (CM)
  – the percent of integration required, compared to integration and test for software of comparable size (IM)

Details of the COCOMO 2.0 Reuse Model can be found in the COCOMO 2.0 documentation

THE APPLICATION COMPOSITION MODEL

1. Estimate the number of Object Points in the system
2. Classify Object Points into simple, medium, and complex categories
3. Apply weighting factors
4. Sum up the total number of Object Points (OPS)
5. Estimate the number of New Object Points (NOPS), taking into account the percent of reuse for the project
6. Apply a productivity factor
7. Compute the estimated person-months
8. Convert to a persons x months estimate
THE COCOMO 2.0 EARLY DESIGN MODEL

- Size is estimated using Unadjusted Function Points (UFP) as the size metric (or other external size measure??)
- Counting is based on the requirements and design documentation
- Counting rules follow the IFPUG guidelines
- Complexity weighting factors (simple, medium, complex) are applied
- Weighted function points are summed up to obtain the UFP count
- the UFP count is converted to thousands of lines of source code (KSLOC) using conversion tables
  - note: the conversion factor should be locally derived

Size-Estim chart 1-71

THE COCOMO 2.0 EARLY DESIGN MODEL - II

- Size, measured in KSLOC, is adjusted by a “breakage” factor:
  \[ AS = S \times [1 + (BRAK/100)] \]
  where
  - AS is adjusted size
  - S is Size in KSLOC obtained by converting UFPs to KSLOC
  and
  - AS is the Adjusted Size
- BRAK accounts for the code that must be discarded because of changing requirements
  - note: “BRAK” is distinct from the %Rework factor of incremental development

Size-Estim chart 1-72
THE POST-ARCHITECTURE MODEL

- Size can be counted in Unadjusted Function Points, Adjusted Function Points, Lines of Code, or any other counting method desired
- Counts are converted to KSLOC
  - size in KSLOC is used in the estimation equation
  - conversion factors must be locally derived, based on past experience

SOME SIZE MEASURES

- Lines of Code
- Pages of Documentation
- Number of Requirements
- Size of Design
- Function Points
- Domain-Specific Measures
- Object Points
SOME SIZE ESTIMATION TECHNIQUES

Size estimation techniques include:

- Analogy
- Rule of Thumb
- Expert Judgment
- The Delphi Approach
- Constrained Estimation
- Statistical Techniques

Any of these techniques can be applied to any of the size measures
- SEE SESSION 2 -
ESTIMATING SOFTWARE SIZE

SESSION 2: SIZE ESTIMATION TECHNIQUES
developed and presented by
Dr. Richard E. Fairley
Professor and Director of Software Engineering
Oregon Graduate Institute
dfairley@cse.ogi.edu

Copyright © 1998 by Richard E. Fairley

SOME SIZE MEASURES

• From Session 1:
  – Lines of Code
  – Pages of Documentation
  – Number of Requirements
  – Size of Design
  – Function Points
  – Domain-Specific Measures
  – Object Points

Copyright © 1998 by Richard E. Fairley
SOME SIZE ESTIMATION TECHNIQUES

- Size estimation techniques include:
  - Analogy
  - Rule of Thumb
  - Expert Judgment
  - The Delphi Approach
  - Constrained Estimation
  - Statistical Techniques

Any of these techniques can be applied to any of the size measures

ESTIMATION BY ANALOGY

- Estimation By Analogy is perhaps the most widely used estimation technique
- Some Examples:
  - we have recently built a similar product containing 300 function points
  - our product is similar to a Pascal compiler of 50 KLOC*
  - our product is similar to an Ada compiler of 300 KLOC

*KLOC: thousands of lines of code
A NOTE ON ESTIMATION-BY-ANALOGY

- Estimation by analogy can be quite sophisticated
  - several attributes of past products can be recorded in a database
  - attributes of the future product can be estimated and matches found for those past products having similar attributes

ESTIMATION USING RULES-OF-THUMB

- Rule-of-Thumb #1: Cellular telephones contain about 60,000 lines of C code
- Rule-of-Thumb #2: In the past, we have built similar systems at a rate of 200 LOC / SM
  - thus, we will need 60,000 / 200 = 300 staff-months of effort
- Rule-of-Thumb #3: Each successive version of our product seems to require 20% more effort than the previous version:
  - thus, we should adjust our estimate upward to 300 x 1.2 = 360 staff-months of effort
SIZE ESTIMATION BY EXPERT JUDGMENT

- Expert judgment involves
  - asking one or more experts
- Advantages of expert judgment
  - experts can account for
    organizational factors, development
    personnel, customer relations, local
    circumstances, political factors, and
    so forth
- Disadvantages of expert judgment
  - experts may be biased
  - experts’ recall may be inexact

DELPHI ESTIMATION

- Delphi estimation involves soliciting estimates from
  several experts in isolation from one another
- The results are summarized by a facilitator and fed
  back to the estimators with a summary of the
  rationale provided by each estimator
- The cycle is repeated until the estimates stabilize
  - the estimates may or may not converge
  - if they don’t converge there are two options:
    1. Wideband Delphi: bring the experts together to
       discuss their estimates and make new
       estimates by “secret ballot”
    2. Use the ranges of estimates to construct a
       probability distribution
SOME SIZE ESTIMATION TECHNIQUES

- Size estimation techniques include:
  - Analogy
  - Rule of Thumb
  - Expert Judgment
  - The Delphi Approach
  => Constrained Estimation
  - Statistical Techniques

CONSTRAINED ESTIMATION

- Estimates are based on available time and money; for example:
  - we have 10 people and 18 months;
  at a productivity rate of 150 LOC/SM we can build 27,000 LOC, for our type of system, using our people and familiar development methods
- We have 6 months to produce a demonstration version of the new system for the trade show
  - we will prioritize the product features and build the most important parts first
SOME SIZE ESTIMATION TECHNIQUES

- Size estimation techniques include:
  - Analogy
  - Rule of Thumb
  - Expert Judgment
  - The Delphi Approach
  - Constrained Estimation

=> Statistical Techniques

STATISTICAL ESTIMATION TECHNIQUES

An estimate should contain:
- a range of estimates for effort, schedule, resources, and defects with associated probabilities
- If we have probability estimates for size, we can use our estimation procedure to calculate probable ranges of effort, schedule, and other factors of interest for various probable sizes
- Statistical size-estimation techniques include:
  - PERT
  - SSM
  - Monte Carlo
PERT ESTIMATION OF SIZE

- The PERT approach to size estimation requires three estimates for each product component:
  - a: the smallest possible size (e.g., 5% probable)
  - m: the most probably size (e.g., 50% probable)
  - b: the largest possible size (e.g., 95% probable)
- These three numbers are used to compute a probability distribution of size for each component
- The size distributions of the components are used to compute an overall size distribution

COMPUTING THE MEANS AND STANDARD DEVIATIONS OF THE PROBABILITY DISTRIBUTIONS

mean: \( E = \frac{(a+4m+b)}{6} \)

std. dev.: \( \sigma = \frac{(b-a)}{3.2} \)

NOTE1: \( \sigma = \frac{(b-a)}{3.2} \) for a wide variety of probability distributions when a = 5% probable; m = 50% probable; and b = 95% probable

NOTE2: Other values of a and b will change the denominator of the \( \sigma \) calculation; for example, \( \sigma \approx 1.5 \) for a = 20% and b = 80%
PERT SIZING

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>m</th>
<th>b</th>
<th>E</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDIT</td>
<td>6k</td>
<td>10k</td>
<td>20k</td>
<td>11.0k</td>
<td>2.3 KLOC</td>
</tr>
<tr>
<td>UPDATE</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>7.5</td>
<td>1.5</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>8</td>
<td>12</td>
<td>19</td>
<td>12.5</td>
<td>1.8</td>
</tr>
<tr>
<td>REPORTS</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>8.0</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- The composite mean, E, is the sum of the individual means: \( E = 39 \) KLOC
- The composite deviation, \( \sigma \), is the RMS value of the individual deviations: \( \sigma = 3.5 \) KLOC

THE CENTRAL LIMIT THEOREM

The Central Limit Theorem states that the means of a collection of independent probability distributions form a normal distribution:

\[ \text{E} \]
\[ \sigma \]
\[ \sigma \]
WBS /PERT SIZING - II

E = 39 KLOC and σ = 3.5 KLOC:

- One Standard Deviation: 68% probable range for the composite mean (most likely) size
  
  \[ E - \sigma = 35.5 \text{ kloc and } E + \sigma = 42.5 \text{ KLOC} \]

- Two Standard Deviations: 97.5% probable
  
  \[ E - 2\sigma = 32 \text{ kloc and } E + 2\sigma = 46 \text{ KLOC} \]

- Three Standard Deviations: 99% probable
  
  \[ E - 3\sigma = 28.5 \text{ KLOC and } E + 3\sigma = 49.5 \text{ KLOC} \]
WHERE DO WE GET THE COMPONENT PROBABILITIES?

- From expert judgment
- From historical data
- From Delphi spread
- From what-if analysis

THE SOFTWARE SIZING MODEL (SSM)

INPUTS:
- SSM uses four inputs for component sizing*
  - Rank order: \(( z, y, x )\)
  - Pair-wise: \(( x, y ); ( z, x ); ( y, z )\)
  - Intervals: 
    
    \[
    \begin{align*}
    50 & < x, y < 100 \\
    100 & < z < 200
    \end{align*}
    \]
  - PERT: \(a, m, and \ b\) for each component
- At least two calibration components

* size can be in lines of code, function points, or any other units of size
SSM OUTPUT

OUTPUTS:

- Expected Size & Standard Deviation For Each Module
- System Size & Confidence Limits

<table>
<thead>
<tr>
<th>MODULE NAME</th>
<th>EXPECTED SIZE</th>
<th>STD DEV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telemetry</td>
<td>12900</td>
<td>2600</td>
</tr>
<tr>
<td>Operations</td>
<td>103700</td>
<td>17300</td>
</tr>
<tr>
<td>Simulator</td>
<td>8500</td>
<td>000 &lt;=</td>
</tr>
<tr>
<td>Monitor</td>
<td>29900</td>
<td>6100</td>
</tr>
<tr>
<td>Status</td>
<td>13400</td>
<td>2000</td>
</tr>
<tr>
<td>Trends</td>
<td>12000</td>
<td>000 &lt;=</td>
</tr>
</tbody>
</table>
SSM OUTPUT (continued)

- **SIZE SUMMARY**
  - Expected System Size: 257000
  - Standard Deviation: 18600

- **CONFIDENCE LIMITS**

<table>
<thead>
<tr>
<th>Probability</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>68%</td>
<td>238400</td>
<td>275600</td>
</tr>
<tr>
<td>95%</td>
<td>219800</td>
<td>294200</td>
</tr>
<tr>
<td>99%</td>
<td>201200</td>
<td>312800</td>
</tr>
</tbody>
</table>

**USING SIZE DISTRIBUTIONS TO COMPUTE EFFORT AND SCHEDULE DISTRIBUTIONS**

- If we have probability distributions for sizes of components, we can:
  1. derive a composite probability distribution of overall size from disparate probability distributions for the components
  then
  3. use selected values of probable size to compute corresponding probable effort and schedule

- Automation of this technique is called Monte Carlo simulation
THE SLIM MODEL

- The Norden-Rayleigh effort equation:
  \[ \text{Effort} = K \left( 1 - e^{-a \times \text{Time}^2} \right) \]

- The Putnam software equation:
  \[ \text{Effort} = \left( \frac{\text{Size}}{\text{Pl}} \right) \times \text{Time}^{-4} \]

- Two equations in two unknowns: Effort and Time
  - solutions are combinations of Effort and Time

- Four parameters: Size, Pl, K, and a
  - Pl, K, and a are determined from past projects
    - used to calibrate SLIM for local use
  - Size, Pl, and MBI (the “a” parameter) are the primary input variables to a calibrated model
SLIM SIZE INPUT

- Size can be expressed in lines of code or function points
- PERT sizing is used
  - smallest, most like, and largest sizes for each software component 1 to 100
- The Productivity Index (Pl) can be specified as known or unknown
  - if known, a single value is used
  - if unknown, a probability distribution is used
- The Manpower Build-up Index (MBI) is similarly specified as known or unknown

MONTE CARLO SIMULATION IN SLIM

- If Pl and MBI are specified as "known," random points are selected from the inverse probability distribution for SLOC and solutions are computed 100 times
- If Pl and MBI are specified as "uncertain," random points are selected from the inverse probability distributions for SLOC, Pl, and MBI and solutions are computed 1000 times
SLIM ESTIMATES

- Estimates of Effort and Time are computed using the Monte Carlo simulation technique
- Values are independently chosen from the Size, PI, and MBI probability distributions
  - the values are used to compute one estimate
  - the process is repeated 1000 times
  - the result is a scatter diagram of Effort and Time combinations

A SLIM EXAMPLE

ESLOC: a = 70; m = 100; b = 130
PI: a = 11; m = 13; b = 15
MBI: a = 1.85; m = 2.0; b = 2.15
AN EXAMPLE (continued)

\[
\begin{align*}
\text{log Effort} & - \text{MBI: } 2 \pm 0.15 \\
\text{M: } 297 \text{ SM} & - \text{\&: } 146 \text{ SM} \\
\text{Pl: } 13 \pm 2 & - \text{SLOC: } 100 \pm 30 \\
\text{log Time} & - \text{M: } 19.1 \text{ MO} \\
& - \text{\&: } 3.2 \text{ MO}
\end{align*}
\]

SCHEDULE ANALYSIS

\[
\begin{align*}
\text{log E} & - \text{MBI: } 2 \pm 15\% \\
\text{Pl: } 13 \pm 2 & - \text{SLOC: } 100 \pm 30 \\
\text{log t} & - \text{Tmax = } 24 \text{ MO} \\
& - 93\% \text{ probable}
\end{align*}
\]
EFFORT ANALYSIS

EMax = 500 SM
87% probable

MBI: 2 ± 15%

PI: 13 ± 2
SLOC: 100 ± 30

log E
log t

Copyright © 1998
by Richard E. Fairley

EFFECT AND SCHEDULE RISK ANALYSIS

EMax = 500 SM
87% probable

MBI: 2 ± 15%

PI: 13 ± 2
SLOC: 100 ± 30

0.87 x 0.93 = 0.81

Tmax = 24 MO
93% probable

Copyright © 1998
by Richard E. Fairley
JOINT PROBABILITY
OF SCHEDULE AND BUDGET

- It is 81% probable the example project can be completed in 24 months with 500 staff months of effort:
  \[0.87 \times 0.93 = 0.81\]

COMPUTING EFFORT AND SCHEDULE DISTRIBUTIONS USING SIZE AND COST DRIVER DISTRIBUTIONS

- If we have probability distributions for sizes of components, we can:
  1. derive a composite probability distribution of overall size from disparate probability distributions for the components
  2. specify probability distributions for the cost drivers
  then
  3. use selected values of probable size and the cost driver probability distributions to compute corresponding probable effort and schedule using a regression-based approach
STEP 1: DERIVE A COMPOSITE PROBABILITY DISTRIBUTION FOR OVERALL SIZE

- Assume three components have size distributions as shown on the following charts.
- (an alternative to STEP 1 is to sample each distribution on each run and use the sum as the size input to the regression equation.)
Probability Density for Size of Component #2

Lognormal distribution
Mean 10.0
Std. Dev. 1.0
Range 9 to 15

Probability Density for Component 2

KLOC 9.00 10.00 15.00

Copyright © 1998 by Richard E. Fairley

Probability Density for Size of Component #3

Triangular distribution
Minimum 4.5
Maximum 15.5
Most Likely 7.0

Probability Density for Component 3

KLOC 4.50 7.00 15.50

Copyright © 1998 by Richard E. Fairley
Probable Size Range of Components 1 + 2 + 3

Forecast: A1
1,000 Trials
Frequency Chart
3 Outliers

Cumulative Probability for Size of Modules 1 + 2 + 3
Cumulative Chart
1,000 Trials
4 Outliers

Copyright © 1998
by Richard E. Fairley

note: this is the composite probability
density, not merely the density of mean values

Size-Estim chart 241

2-21
Probable Size by Percentiles

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>KLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>29.05</td>
</tr>
<tr>
<td>10%</td>
<td>30.00</td>
</tr>
<tr>
<td>20%</td>
<td>30.62</td>
</tr>
<tr>
<td>30%</td>
<td>31.35</td>
</tr>
<tr>
<td>40%</td>
<td>32.04</td>
</tr>
<tr>
<td>50%</td>
<td>32.75</td>
</tr>
<tr>
<td>60%</td>
<td>33.52</td>
</tr>
<tr>
<td>70%</td>
<td>34.59</td>
</tr>
<tr>
<td>80%</td>
<td>35.94</td>
</tr>
<tr>
<td>90%</td>
<td>41.40</td>
</tr>
<tr>
<td>100%</td>
<td>41.40</td>
</tr>
</tbody>
</table>

Copyright © 1998
by Richard E. Fairley

STEP 2: Model the Cost Drivers using Probability Distributions

- See the following charts
STEP 3: Regression-Based Monte Carlo Estimation of Cost (Effort, Schedule, Defects)

- Use the Size and Cost Driver distributions as inputs to a regression equation for example:
  \[ \text{Effort} = 2.5 \times [\text{Size}^{1.25}] \times \text{CPLX} \times \text{PCAP} \]
- Compute a range of probable effort (or cost, schedule, defects)
Estimates vs Commitments

- Commitment = Estimate + Contingency
- Commit to 90th percentile: 267.65 staff-months
  - $2.7M @ $10,000 per staff-month
- Build plans to 70th percentile: 227.27 staff-months
  - $2.3M @ $10,000 per staff-month
- Contingency Reserve = 40 staff-months or $400K
  note: 40 is approx. 17% of 230
CRYSTAL BALL

- Crystal Ball is a Monte Carlo simulation tool that runs as a spreadsheet application
  - probability distributions are specified in “assumptions” cells
  - output equations are programmed in “forecast” cells
  - output distributions are displayed for the forecast cells
- Crystal Ball is marketed by Decisioneering, Inc.
  www.decisioneering.com
  (303)-534-1515

SOME SIZE ESTIMATION TECHNIQUES

Size estimation techniques include:
- Analogy
- Rule of Thumb
- Expert Judgment
- The Delphi Approach
- Constrained Estimation
- Statistical Techniques
  - PERT
  - SSM
  - Monte Carlo

Any of these techniques can be applied to any of the size measures.
AN ESTIMATION PROCEDURE

1. Determine the purpose of and required accuracy for the estimate
2. Identify the required information and sources for it
3. Plan the schedule, resources, and responsibilities for developing the estimate
4. Validate the software requirements
5. Develop a first-cut software architecture
6. Develop a work breakdown structure
7. Estimate the size, scope, complexity, and required quality attributes for each component

Copyright © 1998
by Richard E. Fairley

AN ESTIMATION PROCEDURE - II

8. Supply any additional factors required by the estimation techniques to be used
   – always use more than one estimation technique (for example, a locally calibrated estimation model and WBS-based expert judgment)
9. Prepare estimates using the selected estimation techniques
10. Reconcile differences in the estimates
11. Conduct sensitivity analyses on the estimates
12. Identify risk factors and risk mitigation strategies

Copyright © 1998
by Richard E. Fairley
AN ESTIMATION PROCEDURE - III

13. Prepare a final estimate, including assumptions, confidence level, and contingency plans

14. Prepare a plan for updating the estimate at periodic intervals and on aperiodic events

15. Collect project data and retain a history file

16. Re-estimate cost and schedule in accordance with step 14

A FORMAT FOR DOCUMENTING ESTIMATES

An estimate should contain:
• name(s) of the estimator(s)
• amount of time and effort spent
• the basis of estimation for each method used
• a list of cost drivers and assumptions made
• a range of estimates for effort, schedule, and resources with associated probabilities
• risk factors for the project
• the estimator's level of confidence in the accuracy of the estimate (low, medium, high)
• resources needed to make an improved estimate
SUMMARY

- We must measure some product attributes in order to:
  - compile project histories
  - develop calibrated estimation models
  - make tradeoffs among effort, schedule, product attributes, and quality factors
  - measure productivity, quality, and progress
- Software size correlates with effort, schedule, and resource requirements better than any other product attribute
  - other product attributes can be incorporated as adjustment factors

SUMMARY II: THE FUNDAMENTAL AXIOM OF ESTIMATION

- All estimates are projections from past to future, suitable adjusted to account for differences between past and future
  - the past is captured by historical data relating product attributes to effort, schedule, and resource requirements
  - the future is summarized by the attributes of the product to be built or modified
  - the differences are adjustment factors to account for:
    - different customer
    - different workers
    - different tools
    - different application
    - and so forth
ELEMENTS OF ESTIMATION

Future-Product Attributes

Adjustment Factors

Assumptions and Constraints (=risk factors)

Historical Data

Estimate

Estimation Procedure

SUMMARY - III

Software size measures include:

- Lines of Code
- Pages of Documentation
- Number of Requirements
- Size of Design
- Function Points
- Domain-Specific Measures
- Object Points
SUMMARY - III

Size estimation techniques include:

- Analogy
- Rule of Thumb
- Expert Judgment
- The Delphi Approach
- Constrained Estimation
- Algorithmic Procedures
- Statistical Techniques

Any of these techniques can be applied to any of the size measures

SUMMARY - IV

- The most important points of this presentation:
  1. The Fundamental Axiom
  2. Domain-specific measures of "external size"
  3. The need for uniform counting rules
  4. Using local, relevant historical data and conversion factors
  5. Probabilistic estimation
  6. An estimation procedure
  7. A format for documenting estimates