Technical Metrics for Software
Chapter Outline

- Software Metrics: What and Why?
- 1. How to measure the size of a software?
- 2. How much will it cost to develop a software?
- 3. How many bugs can we expect?
- 4. When can we stop testing?
- 5. When can we release the software?
- 6. What is the complexity of a module?
- 7. What is the module strength and coupling?
Chapter Outline

- 8. What is the reliability at the time of release?
- 9. Which test technique is more effective?
- 10. Are we testing hard or are we testing smart?
- 11. Do we have a strong program or a week test suite?
Technical Metrics

- Are NOT absolute (hence they are open to debate)
- Provide us with a systematic way to assess quality
- Provide insight into product quality on-the-spot rather than after-the-fact
Definitions

- Pressman explained as “A measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of the product or process”.
- Measurement is the act of determine a measure. The metric is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.
- Fenton defined measurement as “it is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules”.
Categories of Metrics

- **Product metrics**: describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.

- **Process metrics**: describe the effectiveness and quality of the processes that produce the software product. Examples are:
  - effort required in the process
  - time to produce the product
  - effectiveness of defect removal during development
  - number of defects found during testing
Project metrics

- describe the project characteristics and execution. Examples are:
  - number of software developers
  - staffing pattern over the life cycle of the software
  - cost and schedule
  - productivity
Software Quality

- Software requirements are the foundation from which quality is measured.
- Specified standards define a set of development criteria that guide the manner in which software is engineered.
- There is a set of implicit requirements that often go unmentioned.
- Software quality is a complex mix of factors that will vary across different applications and the customers who request them.
McCall’s Software Quality Factors

- Maintainability
- Flexibility
- Testability
- Portability
- Reusability
- Interoperability

Product Revision
Product Transition
Product Operation

Correctness  Reliability  Usability  Integrity  Efficiency

\[ F_q = \sum c_i \times m_i \]

Operation curie
McCall’s Quality Parameters

- **Correctness.** The extent to which a program satisfies its specification and fulfills the customer’s mission objectives.

- **Reliability.** The extent to which a program can be expected to perform its intended function with required precision. It should be noted that other, more complete definitions of reliability have been proposed.
<table>
<thead>
<tr>
<th>Quality factor</th>
<th>Correctness</th>
<th>Reliability</th>
<th>Efficiency</th>
<th>Integrity</th>
<th>Maintainability</th>
<th>Flexibility</th>
<th>Testability</th>
<th>Portability</th>
<th>Reusability</th>
<th>Interoperability</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auditability</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Commonality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Complexity</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concision</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consistency</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data commonality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Error tolerance</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution efficiency</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expandability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Hardware Indep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instrumentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modularity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self-documentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simplicity</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>System Indep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traceability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
HP’s FURPS

- **Functionality** - evaluate the feature set and capabilities of the program
- **Usability** - aesthetics, consistency, documentation
- **Reliability** - frequency and severity of failures
- **Performance** - processing speed, response time, resource consumption, throughput, efficiency
- **Supportability** - maintainability testability, compatibility, ease of installation
ISO 9126 Quality Factors

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability
Transition to a Quantitative View

- Previous slides described qualitative factors for the measurement of software quality
- Everyday quality measurements
  - gymnastics, wine tasting, talent contests
  - side by side comparisons
  - quality judged by an expert in the field
- Quantitative metrics don’t explicitly measure quality, but some manifestation of quality
The Challenge of Technical Metrics

- Each quality measurement takes a different view of what quality is and what attributes in a system lead to complexity.
- The goal is to develop measures of different program attributes to use as indicators of quality.
- Unfortunately, a scientific methodology of realizing this goal has not been achieved.
Measurement Principles

- **Formulation** - derivation of software metrics appropriate for the software being considered
- **Collection** - accumulating data required to derive the formulated metrics
- **Analysis** - computation of metrics and application of mathematical tools
- **Interpretation** - evaluation of metrics in an effort to gain insight into the quality of the system
- **Feedback** - recommendations derived from the interpretation of metrics
Attributes of Effective Software Metrics

- Simple and computable
- Empirically and intuitively persuasive
- Consistent and objective
- Consistent in units and dimensions
- Programming language independent
- Effective mechanism for quality feedback
Function Based Metrics

- The Function Point (FP) metric can be used as a means for predicting the size of a system (derived from the analysis model).
  - number of user inputs
  - number of user outputs
  - number of user inquiries
  - number of files
  - number of external interfaces
## Function Point Metric

<table>
<thead>
<tr>
<th>MEASUREMENT PARAMETER</th>
<th>count</th>
<th>simple</th>
<th>average</th>
<th>complex</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of user inputs</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>number of user outputs</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>number of user inquiries</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>number of files</td>
<td>1</td>
<td>7</td>
<td>10</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>number of external interfaces</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>count - total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
</tbody>
</table>

Overall implemented size can be estimated from the projected FP value

\[
FP = \text{count-total} \times (0.65 + 0.01 \times \sum F_i)
\]
The Bang Metric

- Used to predict the application size based on the analysis model.
- The software engineer first evaluates a set of primitives unsubdividable at the analysis level.
- With the evaluation of these primitives, software can be defined as either function-strong or data-strong.
- Once the Bang metric is computed, past history must be used to predict software size and effort.
Different Primitives in Bang Metric

- **Functional primitives (FuP).** The number of transformations (bubbles) that appear at the lowest level of a data flow diagram or which are not further subdivided.

- **Data elements (DE).** The number of attributes of a data object, data elements are not composite data and appear within the data dictionary.

- **Objects (OB).** The number of data objects / Entities.

- **Relationships (RE).** The number of connections between data objects.

- **States (ST).** The number of user observable states in the state transition diagram.

- **Transitions (TR).** The number of state transitions in the state transition diagram.
Additional primitives

- Modified manual function primitives (FuPM). Functions that lie outside the system boundary but must be modified to accommodate the new system.
- Input data elements (DEI).
- Output data elements. (DEO).
- Retained data elements. (DER).
- Data tokens (TCi): Data Objects that exist at boundary of i\(^{th}\) functional primitive.
- Relationship connections (REi). The relationships that connect the \(ith\) object in the data model to other objects.
De Marco Suggestions

$RE/FuP < 0.7$ implies a function-strong application.
$0.8 < RE/FuP < 1.4$ implies a hybrid application.
$RE/FuP > 1.5$ implies a data-strong application.

$TC_{avg} = \sum TC_i / FuP$
Algorithm for Function Strong

set initial value of bang = 0;
do while functional primitives remain to be evaluated
    Compute token-count around the boundary of primitive i
    Compute corrected FuP increment (CFuPI)
    Allocate primitive to class
    Assess class and note assessed weight
    Multiply CFuPI by the assessed weight
    bang = bang + weighted CFuPI
endo
Algorithm for Data Strong

set initial value of bang = 0;
do while Objects remain to be evaluated
    Compute count of relationships for Object i
    Compute corrected OB Increment (COBI)
    bang = bang + COBI
endo
Metrics for Requirements Quality

- Requirements quality metrics - completeness, correctness, understandability, verifiability, consistency, achievability, traceability, modifiability, precision, and reusability - design metric for each.

- E.g., let $n_r = n_f + n_{nf}$, where
  - $n_r =$ number of requirements
  - $n_f =$ number of functional requirements
  - $n_{nf} =$ number of nonfunctional requirements
Metrics for Requirements Quality

- **Specificity (lack of ambiguity)**
  - \( Q = \frac{n_{ui}}{n_r} \)
  - \( n_{ui} \) - number of requirements for which all reviewers had identical interpretations

- **For completeness,**
  - \( Q = \frac{n_u}{(n_i \times n_s)} \)
  - \( n_u \) = number of unique function requirements
  - \( n_i \) = number of inputs specified
  - \( n_s \) = number of states specified
High-Level Design Metrics

- By Card & Glass in 1990

- Structural Complexity
  - $S(i) = f_{out}^2(i)$
  - $f_{out}(i) =$ fan-out of module $i$

- Data Complexity
  - $D(i) = \frac{v(i)}{f_{out}(i) + 1}$
  - $v(i) =$ # of input and output variables to and from module $i$

- System Complexity
  - $C(i) = S(i) + D(i)$
High-Level Design Metrics (Cont.)

- By Fenton 1991

**Morphology Metrics**

- size = \( n + a \)
- \( n \) = number of modules
- \( a \) = number of arcs (lines of control)
- arc-to-node ratio, \( r = a/n \)
- depth = longest path from the root to a leaf
- width = maximum number of nodes at any level
Morphology Metrics

- size
- depth
- width
- arc-to node ratio
AF Design Structure Quality Index

- $S_1 =$ total number of modules
- $S_2 =$ # modules dependent upon correct data source or produces data used elsewhere, excl. control
- $S_3 =$ # modules dependent upon prior processing
- $S_4 =$ total number of database items
- $S_5 =$ # unique database items
- $S_6 =$ # of database segments
- $S_7 =$ # modules with single entry & exit
Air Force Design Structure Quality Index

- $D_1 = 1$ if arch design method used, else 0
- $D_2 = 1 - \frac{S2}{S1}$ -- module independence
- $D_3 = 1 - \frac{S3}{S1}$ -- independence of prior processing
- $D_4 = 1 - \frac{S5}{S4}$ -- database size
- $D_5 = 1 - \frac{S6}{S4}$ -- DB compartmentalization
- $D_6 = 1 - \frac{S7}{S1}$ -- Module entrance/exit
AF Design Structure Quality Index

- DSQI = $\Sigma w_i D_i$, where the $w_i$ are weights totaling 1 which give the relative importance.
- The closer this is to one, the higher the quality.
- This is best used on a comparison basis, i.e., with previous successful projects.
- If the value is too low, more design work should be done.
Component-Level Design Metrics

- Cohesion Metrics
- Coupling Metrics
  - data and control flow coupling
  - global coupling
  - environmental coupling
- Complexity Metrics
  - Cyclomatic complexity
  - Experience shows that if this $> 10$, it is very difficult to test
Cohesion Metrics

- **Data slice** - data values *within the module* that affect the module location at which a backward trace began.
- **Data tokens** - Variables defined for a module
- **Glue Tokens** - The set of tokens lying on multiple data slices
- **Superglue tokens** - The set of tokens on all slices
- **Stickiness** - of a glue token is proportional to number of data slices that it binds

**Strong Functional Cohesion**

\[
SFC(i) = \frac{SG(i)}{\text{tokens}(i)}
\]
Coupling Metrics

- **Data and control flow coupling**
  - \( d_i = \) number of input data parameters
  - \( c_i = \) number of input control parameters
  - \( d_0 = \) number of output data parameters
  - \( c_0 = \) number of output control parameters

- **Global coupling**
  - \( g_d = \) number of global variables used as data
  - \( g_c = \) number of global variables used as control

- **Environmental coupling**
  - \( w = \) number of modules called (fan-out)
  - \( r = \) number of modules calling the module under consideration (fan-in)
  
  **Module Coupling:**  
  \[ m_c = \frac{1}{d_i + 2c_i + d_0 + 2c_0 + g_d + 2g_c + w + r} \]
  
  - \( m_c = \frac{1}{1 + 0 + 1 + 0 + 0 + 1 + 0} = .33 \) (Low Coupling)
  - \( m_c = \frac{1}{5 + 2*5 + 5 + 2*5 + 10 + 0 + 3 + 4} = .02 \) (High Coupling)
Interface Design Metrics

- Layout Entities - graphic icons, text, menus, windows, etc.
- Layout Appropriateness
  - absolute and relative position of each layout entity
  - frequency used
  - cost of transition from one entity to another
- LA = 100 x [(cost of LA-optimal layout) / (cost of proposed layout)]
- Final GUI design should be based on user feedback on GUI prototypes
Metrics for Source Code

- **Software Science Primitives**
  - \( n_1 = \) the number of distinct operators
  - \( n_2 = \) the number of distinct operands
  - \( N_1 = \) the total number of operator occurrences
  - \( N_2 = \) the total number of operand occurrences

- **Length:** \( N = n_1 \log_2 n_1 + n_2 \log_2 n_2 \)
- **Volume:** \( V = N \log_2 (n_1 + n_2) \)
SUBROUTINE SORT (X,N)
DIMENSION X(N)
IF (N.LT.2) RETURN
DO 20 I=2,N
   DO 10 J=1,I
      IF (X(I).GE.X(J) GO TO 10
         SAVE = X(I)
         X(I) = X(J)
         X(J) = SAVE
   10 CONTINUE
20 CONTINUE
RETURN
END

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>END OF STATEMENT</td>
<td>7</td>
</tr>
<tr>
<td>ARRAY SUBSCRIPT</td>
<td>6</td>
</tr>
<tr>
<td>=</td>
<td>5</td>
</tr>
<tr>
<td>IF( )</td>
<td>2</td>
</tr>
<tr>
<td>DO</td>
<td>2</td>
</tr>
<tr>
<td>,</td>
<td>2</td>
</tr>
<tr>
<td>END OF PROGRAM</td>
<td>1</td>
</tr>
<tr>
<td>.LT.</td>
<td>1</td>
</tr>
<tr>
<td>.GE.</td>
<td>1</td>
</tr>
<tr>
<td>GO TO 10</td>
<td>1</td>
</tr>
</tbody>
</table>

\( n_1 = 10 \) \quad N_1 = 28
\( n_2 = 7 \) \quad N_2 = 22
Metrics for Testing

- Analysis, design, and code metrics guide the design and execution of test cases.

Metrics for Testing Completeness

- Breadth of Testing - total number of requirements that have been tested
- Depth of Testing - percentage of independent basis paths covered by testing versus total number of basis paths in the program.
- Fault profiles used to prioritize and categorize errors uncovered.
Metrics for Maintenance

Software Maturity Index (SMI)

- \( M_T = \) number of modules in the current release
- \( F_c = \) number of modules in the current release that have been changed
- \( F_a = \) number of modules in the current release that have been added
- \( F_d = \) number of modules from the preceding release that were deleted in the current release

\[
SMI = \frac{[M_T - (F_c + F_a + F_d)]}{M_T}
\]
Summary

- Software metrics provide a quantitative way to assess the quality of product attributes.
- A software metric needs to be simple, computable, persuasive, consistent, and objective.
- The function point and bang metrics provide quantitative means for evaluating the analysis model.
- Metrics for design consider high-level, component level, and interface design issues.
Summary

- Interface design metrics provide an indication of layout appropriateness for a GUI.
- Using the number of operators and operands present in the code provides a variety of metrics to assess program quality.
- Using the metrics as a comparison with known successful or unsuccessful projects is better than treating them as absolute quantities.