Change Models and Metrics

Change Data Classes

Changes can be categorized by

- **purpose**: e.g., enhancement, adaptive, corrective, preventive
- **type**: e.g., requirements, specification, design, architecture, planned enhancements, insert/delete debug code, improve clarity, optimize: space or time, feature, enhancement, bug
- **cause**: e.g., market/external and internal needs
- **size**: e.g., number of lines of code, number of components affected
- **disposition**: e.g., rejected as a change, not relevant, under consideration, being worked on, completed, saved for next enhancement
- **level of document changed**: e.g., changes back to requirements document
- **number of customers affected**: e.g., affects certain customer classes

Sample Change Metrics

- number of enhancements per month
- number of changes per line of code
- number of changes during requirements
- number of changes generated by the user vs. internal
- number of changes rejected: total number of changes
- Change Report history profile

Defect Definitions

- **Errors**: Defects in the human thought process made while trying to understand given information, to solve problems, or to use methods and tools
  - One error may cause several faults
  - Various errors may cause identical faults

- **Faults**: Concrete manifestations of errors within the software
  - A particular failure may be caused by several faults
  - Some faults may never cause a failure (difference between reliability and correctness)

Building Models

Methods and Tools Dealing with Software Defects

- **Problem**
- **Understand**
- **Analyse**
- **Executive**
- **Document**
- **Manage**

Methods and Tools Dealing with Defects

- **Prevent**
- **Isolate**
- **Detect**
- **Errors**
- **Faults**
- **Failures**
Combining Models

Change and Defect Models and Metrics

Fault Data Classes
- Fault detection time: the phase or activity in which the fault was detected.
  - Example subclasses: requirements, specification, design, code, unit test, system test, acceptance test, maintenance
- Fault Density: number of faults per KLOC
- Effort to Isolate/Fix: time taken to isolate and fix a fault usually in time intervals
  - Example subclasses: 1 hour or less, 1 hour to 1 day, 1 day to 3 days, more than 3 days
- Omission/commission: where omission is neglecting to include some entity and commission is the inclusion of some incorrect executable statement or fact
- Algorithmic fault: the problem with the algorithm
  - Example subclasses: control flow, interface, data <definition, initialization, use>.

Failure Data Classes
- Failure detection time: the phase or activity in which the failure was detected
  - Example subclasses: unit test, system test, acceptance test, operation
- System Severity: the level of effect the failure has on the system
  - Example subclasses: operation stops completely, operation is significantly impacted, prevents full use of features but can be compensated, minor or cosmetic
- Customer Impact: the level of effect the failure has on the customer
  - Example subclasses: usually similar to the subclasses for system severity but may be categorized differently because of subjective implications and customer satisfaction issues

Sample Defect Metrics
- Number of faults per line of code
- Number of faults discovered during system test, acceptance test and one month, six months, one year after system release
- Ratio of faults in system test on this project to faults found after system test
- Number of severity 1 failures that are caused by faults of omission
- Percent of failures found during system test
- Percent of interface faults found by code reading

Process Models and Metrics
**PROCESS MODELS AND METRICS**

### Process Data

These metrics capture information about the processes being performed. They can be associated with the definition of the process, process performance, and the process definition. Process metrics can be used to evaluate the process, gain insight into the product, and provide information that helps tailor and evolve the processes over time.

### PROCESS METRICS

**Building Process Models**

Modeling the education and training process

<table>
<thead>
<tr>
<th>Process:</th>
<th>1. provide the individual with training manuals they must read.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. provide a course, educating the individual in the process.</td>
</tr>
<tr>
<td></td>
<td>3. provide training by applying the process to a toy problem.</td>
</tr>
<tr>
<td></td>
<td>4. assign the individual to a project that is using the process,</td>
</tr>
<tr>
<td></td>
<td>mentored by an experienced method user.</td>
</tr>
<tr>
<td></td>
<td>5. after the individual is considered fully trained in the</td>
</tr>
<tr>
<td></td>
<td>process.</td>
</tr>
</tbody>
</table>

Convert to an operational model by associating interval values with the process steps. Here, each step represents a further passage along the interval scale. Thus, a value of

- 0 implies no training,
- 1 implies the individual has read the manuals,
- 2 implies the individual has been through a training course,
- 3 implies the individual has had experience in a laboratory environment,
- 4 implies the process was used on a project before; under tutelage, and
- 5 implies the process has been used on several projects.

### PROCESS METRICS

**Defining Process Metrics**

Measuring the education and training process

If the education and training process model is valid, the subjective rating is valid. Use the GQM to generate a question that gathers the information for the model. Characterize the process experience of the team (subjective rating per person):

- 0: none
- 1: have read the manuals
- 2: have had a training course
- 3: have had experience in a laboratory environment
- 4: have used on a project before
- 5: have used on several projects before

Build an interpretation model for a team of developers: e.g., a minimum requirement is that all team members have at least a three and the team leader has a five.

This evaluation process will become more effective with experience over time.

### PRODUCT METRICS

**Product Data**

These metrics depend on characteristics of the product. These product characteristics can be:

- logical, e.g., application domain, function
- physical, e.g., size, structure
- dynamic, e.g., MTTF, test coverage
- use and context related, e.g., design method used to develop

Product metrics can be used to:

- evaluate the process or the product
- estimate the cost of quality of the product
- monitor the stability or quality of the product over time.

### PRODUCT METRICS

**Size Metrics**

There are an enormous number of size metrics, the most common for being:

- lines of code
- function points
- pages of documentation
- modules
- operators and operands

Size metrics can be used accurately at different points in time. Lines of code is accurate after the fact but can be estimated function points can be calculated based upon the specification.

Size metrics can be used to characterize the product, evaluate the effect of some treatment variable, such as a process predict some other variable, such as cost.

<table>
<thead>
<tr>
<th>Size Metrics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>We will divide the static product metrics into three basic classes</td>
<td></td>
</tr>
<tr>
<td>Size metrics</td>
<td></td>
</tr>
<tr>
<td>Control Structure Metrics</td>
<td></td>
</tr>
<tr>
<td>Data Structure Metrics</td>
<td></td>
</tr>
<tr>
<td>Size metrics attempt to measure the physical size of the product</td>
<td></td>
</tr>
<tr>
<td>Control structure metrics attempt to measure the control flow of the product</td>
<td></td>
</tr>
<tr>
<td>Data structure metrics attempt to measure the data interaction of the product</td>
<td></td>
</tr>
<tr>
<td>There are mixes of these metrics, e.g., that deal with the interaction between control and data flow</td>
<td></td>
</tr>
<tr>
<td>Dynamic metrics can be viewed as checking on the behavior of the input to the code, e.g., coverage metrics</td>
<td></td>
</tr>
<tr>
<td>Behavior of the code, e.g., reliability metrics</td>
<td></td>
</tr>
<tr>
<td>There are an enormous number of size metrics, the most common for being</td>
<td></td>
</tr>
<tr>
<td>lines of code</td>
<td></td>
</tr>
<tr>
<td>function points</td>
<td></td>
</tr>
<tr>
<td>pages of documentation</td>
<td></td>
</tr>
<tr>
<td>modules</td>
<td></td>
</tr>
<tr>
<td>operators and operands</td>
<td></td>
</tr>
<tr>
<td>Size metrics can be used accurately at different points in time</td>
<td></td>
</tr>
<tr>
<td>lines of code is accurate after the fact but can be estimated function points can be calculated based upon the specification</td>
<td></td>
</tr>
<tr>
<td>Size metrics can be used to characterize the product</td>
<td></td>
</tr>
<tr>
<td>evaluate the effect of some treatment variable, such as a process</td>
<td></td>
</tr>
<tr>
<td>predict some other variable, such as cost</td>
<td></td>
</tr>
</tbody>
</table>
**Product Metrics**

**Lines of Code Metrics**

Lines of code can be counted as:
- all source lines
- all non-blank source lines
- all non-blank, non-commentary source lines
- all semi-colons
- all executable statements

The definition depends on the use of the metric, e.g.,
- to estimate effort we might use all source lines as they all take effort
- to estimate functionality we might use all executable statements as they come closest to representing the amount of function in the system

Lines of code vary with the language being used
have proven to be the most common, durable, cheapest metric to calculate
are most used to characterize the product and predict effort

**Function Points**

A function point is an attempt to capture the interfaces of a system as a means of estimating functionality, size, effort

It can be applied in the early phases of a project (requirements, preliminary design)

A function point is a specific user functionality delivered by the application

- Input type, e.g., screen data, menu selection
- Output Type, e.g., report, transferred data, message
- Query type, e.g., request/retrieval combination
- File type, e.g., database/record, indexed file
- External Interface, e.g., reference data, external data bases

**Function Points Calculation**

**Complexity Weights**

<table>
<thead>
<tr>
<th></th>
<th>SIMPLE</th>
<th>AVERAGE</th>
<th>COMPLEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Output</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Query</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_Input part</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>_Output part</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>File</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>External</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

The original function point approach was proposed by Albrecht in the late 70s in the IBM Data Processing Division

There is currently an International Function Point Users Group (IFPUG) whose mission is to coordinate the state of the practice, support users and standardize the approach

Function Point Counting Practices Manual (Version 4)

**Function Points Calculation**

**Application Characteristics**

<table>
<thead>
<tr>
<th>DATA COMMUNICATIONS</th>
<th>ON-LINE UPDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTRIBUTED DATA OR PROCESSING</td>
<td>COMPLEX PROCESSING</td>
</tr>
<tr>
<td>HEAVILY-USED CONFIGURATION</td>
<td>REUSABILITY</td>
</tr>
<tr>
<td>TRANSACTION RATE</td>
<td>CONVERSION &amp; INSTALLATION EASE</td>
</tr>
<tr>
<td>END USER EFFICIENCY</td>
<td>OPERATIONAL EASE</td>
</tr>
<tr>
<td>END USER EFFICIENCY</td>
<td>MULTIPLE-SITE</td>
</tr>
</tbody>
</table>

**Influence Scale**

| 5 | STRONG THROUGHOUT |
| 4 | SIGNIFICANT |
| 3 | AVERAGE |
| 2 | MODERATE |
| 1 | INSIGNIFICANT, MINOR |
| 0 | NONE |

\[ \text{FUNCTION POINTS} = (\sum \text{INPUTS} \times \text{WEIGHTS} + \sum \text{OUTPUTS} \times \text{WEIGHTS}) + \sum \text{QUERIES} \times \text{WEIGHTS} + \sum \text{FILES} \times \text{WEIGHTS} + \sum \text{INTERFACES} \times \text{WEIGHTS}) \times (0.65 + 1\% \text{ TOTAL INFLUENCE}) \]
Function Points Calculation

**Example**

**SPELLING CHECKER SPECIFICATION:** The checker accepts as input a document file and an optional personal dictionary file. The checker lists all words not contained in either the dictionary or the personal dictionary file. The user can query the number of words processed and the number of spelling 'errors' found at any stage during the processing.

**DEVIATION PROJECTS WITHIN RANGE**

<table>
<thead>
<tr>
<th>DEVIATION</th>
<th>NUMBER</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>± 15%</td>
<td>9</td>
<td>33%</td>
</tr>
<tr>
<td>± 25%</td>
<td>12</td>
<td>37%</td>
</tr>
<tr>
<td>± 25%</td>
<td>17</td>
<td>51%</td>
</tr>
</tbody>
</table>

Software Science

**MEASURABLE PROPERTIES OF ALGORITHMS**

- \( n_k \): Unique or distinct operations in an implementation
- \( n_0 \): Unique or distinct operations in a program
- \( n_k \): Total usage of all operators
- \( f_j \): Frequency of the \( j \) most frequent operator

The implementation length is

\[
N = n + n_k
\]

and

\[
X = \sum_{j=1}^{n_k} x_j^j
\]
**Example: Euclid's Algorithm**

```
IF (A = O)
LAST:
BEGIN GCD := B; RETURN END;
IF (B = O)
BEGIN GCD := A; RETURN END;
HERE:
Q := A/B; R := A - B X Q;
IF (R = O) GO TO LAST;
A := B; B := R; GO TO HERE
```

**Operator Parameters**

**Greatest Common Divisor Algorithm**

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>j</th>
<th>t_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>;</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>:=</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>( ) OR BEGIN...END</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>IF</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>=</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>/</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>GO TO HERE</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>GO TO LAST</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

n_1 = 10    N_2 = 31

**Operand Parameters**

**Greatest Common Divisor Algorithm**

<table>
<thead>
<tr>
<th>OPERAND</th>
<th>j</th>
<th>t_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>R</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>GCD</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

n_2 = 6    N_2 = 21

**Software Science Metrics**

**PROGRAM LENGTH:**

n_2 ~ \hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2

\hat{N} = The number of bits necessary to represent all things that exist in the program at least once

n = The number of bits necessary to represent the symbol table

**PROGRAM VOLUME:** (Size of an implementation)

V = N \log_2 n

B = The number of bits necessary to represent the program

**Software Science Majors**

**PROGRAMMING EFFORT:**

E = V D = V/L = V/V

E = The effort required to comprehend an implementation rather than produce it

E = A measure of program clarity

**TIME:**

T = E/S = V/S = V/V

T = the time to develop an algorithm

**ESTIMATED BUGS:**

\hat{B} = (L E) / E

Where E = The mean effort between potential errors in programming

\hat{B} = the number of errors expected in a program

**POTENTIAL VOLUME:**

V = (2 + n_2) \log_2 (2 + n_2)

Where n_2 represents the number of input/output parameters

V = A measure of the specification for an algorithm

**PROGRAM LEVEL:** (Level of an implementation)

L = V/V

L = 2n_2

L = Difficult

n_2 N_2

D = L / S L = Difficulty

```

---

**Software Science Metrics**

**PROGRAM LEVEL:** (Level of an implementation)

L = V/V

L = 2n_2

L = Difficult

n_2 N_2

D = L / S L = Difficulty

---

**Software Science Majors**

**PROGRAMMING EFFORT:**

E = V D = V/L = V/V

E = The effort required to comprehend an implementation rather than produce it

E = A measure of program clarity

**TIME:**

T = E/S = V/S = V/V

T = the time to develop an algorithm

**ESTIMATED BUGS:**

\hat{B} = (L E) / E

Where E = The mean effort between potential errors in programming

\hat{B} = the number of errors expected in a program

---

**Example: Euclid's Algorithm**

```
IF (A = O)
LAST:
BEGIN GCD := B; RETURN END;
IF (B = O)
BEGIN GCD := A; RETURN END;
HERE:
Q := A/B; R := A - B X Q;
IF (R = O) GO TO LAST;
A := B; B := R; GO TO HERE
```
Cyclomatic Complexity

- The **cyclomatic number** \( V(G) \) of a graph \( G \) with \( n \) vertices, \( e \) edges, and \( p \) connected components is:
  \[ V(G) = e - n + p \]
- In a strongly connected graph \( G \), the cyclomatic number is equal to the maximum number of linearly independent circuits.

### Simplification

\[ \Theta = \# \text{Function Nodes} \]
\[ \Pi = \# \text{Predicate Nodes} \]
\[ \lambda = \# \text{Collecting Nodes} \]

Then:
\[ e = 1 + \Theta + 3\lambda \]
\[ n = \Theta + 2\lambda + 2 \]

Assuming \( p = 1 \) and \( v = e - n + 2p \) yields
\[ v = (1 + \Theta + 3\lambda) - (\Theta + 2\lambda + 2) + 2 = \Theta + 1 \]

Properties of Cyclomatic Complexity

1. \( V(G) \geq 1 \)
2. \( V(G) = 1 \) if and only if \( G \) is a linearly independent path in \( G \), it is the size of a basis set.
3. Inserting or deleting functional statements to \( G \) does not affect \( V(G) \).
4. \( G \) has only one path if and only if \( V(G) = 1 \).
5. Inserting a new edge in \( G \) increases \( V(G) \) by 1.
6. \( V(G) \) depends only on the decision structure of \( G \).
   - More than 1 component
   - \( V(M \cup A \cup B) = e - n + 2p = 13 - 13 + 2(3) = 6 \)
   - For a collection of components
     \[ V(G) = \sum V(C_i) \]

SEL

**Evaluating and Comparing Software Metrics**

**Goals**

- Do measures like Cyclomatic Complexity and the Software Science Metrics relate to effort and quality?
- Does the correspondence increase with greater accuracy of data reporting?
- How do these metrics compare with traditional size metrics such as number of source lines or executable statements?
- How do these metrics relate to one another?

**Definitions**

- **Effort**: The number of man-hours programmers and managers spent from the beginning of functional design to the end of acceptance testing.
- **Quality**: The number of program failures reported during the development of the product.

**Metric Evaluation in the SEL**

**Size and Complexity Measures Investigated**

**The Data:**

- **Commercial Software**: Satellite Ground Support
- Systems consist of 51,000 to 112,000 lines of FORTRAN source code.
- Ten to sixty-one percent of source code modified from previous projects.
- Development effort ranges from 6900 to 22,300 man-hours.

**This Analysis Focuses On:**

- Data from seven projects.
- Only newly developed modules (i.e., subroutines, functions, main procedures and block data).
PRODUCT METRICS

Coverage Metrics

Based upon checking what aspects of the product are affected by a set of inputs.

For example:
- procedure coverage - which procedures are covered by the set of inputs
- statement coverage - which statements are covered by the set of inputs
- branch coverage - which parts of a decision node are covered by the set of inputs
- path coverage - which paths are covered by the set of inputs
- requirements section coverage - which parts of the requirements document have been read

Used to:
- check the quality of a test suite
- support the generation of new test cases