There are a large number of product types: requirements documents, specifications, design, code, specific components, test plans, ...

There are many abstractions of these products that depend on different characteristics:
- logical, e.g., application domain, function
- static, e.g. size, structure
- dynamic, e.g., MTTF, test coverage
- use and context related, e.g., design method used to develop

Product models and 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 Models and Metrics

Logical Characteristics

The logical characteristic application can be measured on a nominal scale:
- flight software, ground support software, ...

The logical characteristic function can be represented as
- a mathematical function abstraction for a program, e.g., y = f(x), a state function abstraction for a module
- a nominal class, e.g., the component represents a mathematical function, data structure, ...

Product Models and Metrics

Static Characteristics

We can divide the static product the characteristics into three basic classes:
- Size
- Structure, e.g.,
  - Control Structure
  - Data Structure

Size attempts to model and measure the physical size of the product

Structure models and metrics attempt to capture some aspect of the physical structure of the product, e.g.,
- Control structure metrics measure the control flow of the product
- Data structure metrics measure the data interaction of the product

There are mixes of these metrics, e.g., that deal with the interaction between control and data flow.
Product Models and Metrics

Size

There are many size models and metrics, depending on the product, e.g.,
- source code: lines of code, number of modules
- executables: space requirements, lines of code
- specification: function points
- requirements: number of requirements, 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 are often used to
- characterize the product
- evaluate the effect of some treatment variable, such as a process
- predict some other variable, such as cost

Product Models and Metrics

Lines of Code Metrics

Lines of code can be measured 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
- are the most common, durable, cheapest metric to calculate
- are most often used to characterize the product and predict effort
Product Models and Metrics

Function Points

One model of a product is to view it as a set of interfaces, e.g., files, data passed, etc.

If a system is primarily transaction processing and the “bulk” of the system deals with transformations on files, this is a reasonable view of size.

Function Points were originally suggested as a measure of size by Al Albrecht at IBM, a means of estimating functionality, size, effort.

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

Albrecht

Product Models and Metrics

Function Points

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

It differentiates five types of files or data:

- 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
Product Models and Metrics

Function Points

There are counting rules:

Only user requested and visible components are counted

Components such as internally maintained data entries, externally maintained data entries, data maintenance activities, data output and data retrieval are categorized and valued

The final count is adjusted based upon the general characteristics of the system (distributed functions, performance considerations, complex processing)

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 User Group (IFPUG) whose mission is to coordinate that the state of the practice, support users and standardize the approach

Function Point Counting Practices Manual (Version 4)

Function Points Calculation

Complexity Weights

<table>
<thead>
<tr>
<th></th>
<th>SIMPLE</th>
<th>AVERAGE</th>
<th>COMPLEX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td><strong>Output type</strong></td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td><strong>Query type</strong></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><strong>File type</strong></td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td><strong>External interface</strong></td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>
Function Points Calculation

Application Characteristics

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Influence Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA COMMUNICATIONS</td>
<td>0 N/A</td>
</tr>
<tr>
<td>DISTRIBUTED DATA OR PROCESSING</td>
<td>0 N/A</td>
</tr>
<tr>
<td>PERFORMANCE OBJECTIVES</td>
<td>0 N/A</td>
</tr>
<tr>
<td>HEAVILY-USED CONFIGURATION</td>
<td>0 N/A</td>
</tr>
<tr>
<td>TRANSACTION RATE</td>
<td>0 N/A</td>
</tr>
<tr>
<td>ON-LINE DATA ENTRY</td>
<td>0 N/A</td>
</tr>
<tr>
<td>END USER EFFICIENCY</td>
<td>0 N/A</td>
</tr>
<tr>
<td></td>
<td>1 INFLUENCE SCALE</td>
</tr>
<tr>
<td></td>
<td>2 INFLUENCE SCALE</td>
</tr>
<tr>
<td></td>
<td>3 INFLUENCE SCALE</td>
</tr>
<tr>
<td></td>
<td>4 INFLUENCE SCALE</td>
</tr>
<tr>
<td></td>
<td>5 INFLUENCE SCALE</td>
</tr>
</tbody>
</table>

FUNCTION POINTS =

\[
\text{FUNCTION POINTS} = \left( \sum \text{INPUTS} \times \text{WEIGHTS} + \sum \text{OUTPUTS} \times \text{WEIGHTS} + \sum \text{QUERIES} \times \text{WEIGHTS} + \sum \text{FILES} \times \text{WEIGHTS} + \sum \text{INTERFACES} \times \text{WEIGHTS} \right) \times (0.65 + 1\% \text{ TOTAL INFLUENCE})
\]
**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 files. The user can query the number of words processed and the number of spelling ‘errors’ found at any stage during the processing.

```
+-----------------+-----------------+-----------------+
| USER            | SPELLING        | DICTIONARY      |
|                 | CHECKER         |                 |
+-----------------+-----------------+-----------------+
| Document file   |                 |                 |
| Personal dictionary |               |                 |
| Misspelt words report |         |                 |
| Errors inquiry/# errors |     |                 |
| Words processed inquiry/# words | |                 |
+-----------------+-----------------+-----------------+

ASSUMING AVERAGE COMPLEXITY IN EACH CASE
AND MINOR IMPACT

<table>
<thead>
<tr>
<th>INPUTS</th>
<th>OUTPUTS</th>
<th>QUERIES</th>
<th>FILES</th>
<th>INTERFACES</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>15</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>18</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>14</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

65 \*(0.65 + 0.01*14) = 51.35 FUNCTION POINTS
**FUNCTION POINTS**

**Estimating Costs**

- Cost Estimation using Function Points requires
  - Cost or effort data for previous projects
  - Function Points counted in previous projects
- The estimation process
  - The cost (or effort) data of previous projects is plotted against the Function Points counted in those projects
  - This curve is used to derive the cost of the current project from the value of its Function Points

**Example:**

\[ \text{Effort (Days) = } 2.37 \times \text{(Function Points)} + 41.09 \]

**FUNCTION POINTS**

**Assessing Productivity**

- Productivity assessment using Function Points requires
  - productivity figures for previous projects
  - Function Points counted in previous projects
- The assessment process:
  - Data about the productivity in previous projects is plotted against the FP count in those projects
  - The expected productivity is the productivity value for the FPs of this project
  - Discrepancies between actuals and expected are analyzed
FUNCTION POINTS
Reliability Of Function Point Based Measures

• Generally a productivity model is considered good if it is capable of giving an estimate with 25% accuracy in 75% of the cases
• Studies conducted in MIS (Management Information Systems) environments show that, for both development and maintenance, Function Points based measures often satisfy the criterion
• Example: Canadian financial institution study on maintenance activities (21 projects, 332 average staff days per project, min 52, max 532)

<table>
<thead>
<tr>
<th>DEVIATION</th>
<th>PROJECTS WITHIN RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>+ / - 10%</td>
<td>9</td>
</tr>
<tr>
<td>+ / - 20%</td>
<td>12</td>
</tr>
<tr>
<td>+ / - 26%</td>
<td>17</td>
</tr>
</tbody>
</table>

Software Science

Suppose we view a module or program as an encoding of an algorithm and seek some minimal coding of its functionality

The model would be an abstraction of the smallest number of operators and operands (variables) necessary to compute a similar function

And then the smallest number of bits necessary to encode those primitive operators and operands

This model was proposed by Maurice Halstead as a means of approximating program size.
Software Science

- MEASURABLE PROPERTIES OF ALGORITHMS

\[ n_1 = \text{# Unique or distinct operators in an implementation} \]
\[ n_2 = \text{# Unique or distinct operands in an implementation} \]
\[ N_1 = \text{# Total usage of all operators} \]
\[ N_2 = \text{# Total usage of all operands} \]
\[ f_{1,j} = \text{# Occurrences of the } j\text{th most frequent operator} \quad j = 1, 2, \ldots n_1 \]
\[ f_{2,j} = \text{# Occurrences of the } j\text{th most frequent operand} \quad j = 1, 2, \ldots n_2 \]

THE VOCABULARY \( n \) IS \( n = n_1 + n_2 \)

THE IMPLEMENTATION LENGTH IS \( N = N_1 + N_2 \)

and \( N_1 = \sum_{j=1}^{n_1} f_{1,j} \)
\( N_2 = \sum_{j=1}^{n_2} f_{2,j} \)
\( N = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} \)

Example: Euclid’s Algorithm

\[ \text{IF } (A = O) \]
\[ \text{LAST: BEGIN GCD := B; RETURN END;} \]
\[ \text{IF } (B = O) \]
\[ \text{BEGIN GCD := A; RETURN END;} \]
\[ \text{HERE: } G := A/B; \quad R := A - B \times G; \]
\[ \text{IF } (R = O) \text{ GO TO LAST;} \]
\[ \text{A := B; B := R; GO TO HERE} \]
**Operator Parameters**

**Greatest Common Divisor Algorithm**

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>j</th>
<th>f_{1j}</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>
<tr>
<td>[n_1 = 10 \quad N_1 = 31]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operand Parameters**

**Greatest Common Divisor Algorithm**

<table>
<thead>
<tr>
<th>OPERAND</th>
<th>j</th>
<th>f_{2j}</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>O</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>
<tr>
<td>[n_2 = 6 \quad N_2 = 21]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Software Science Metrics

**PROGRAM LENGTH:**
\[ N \sim \hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2 \]

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

**PROGRAM VOLUME:** (Size of an implementation)
\[ V = N \log_2 n \]

\( V \) = The number of bits necessary to represent the program

**POTENTIAL VOLUME:** (Minimal size of an implementation)
\[ V^* = (2 + n^*_2) \log_2 (2 + n^*_2) \]

Where \( n^*_2 \) represents the number of input/output parameters

\( V^* \) = A measure of the specification for an algorithm

---

**Software Science Metrics**

**PROGRAM LEVEL:** (Level of an implementation)
\[ L = V^* / V \]

\( D = 1/L \) = Difficulty

**PROGRAMMING EFFORT:**
\[ E = V \cdot D = V/L = V/V \]

\( E \) = The effort required to comprehend an implementation rather than produce it
\( E \) = A measure of program clarity
The **Cyclomatic Number** $V(G)$ of a graph $G$ with $n$ vertices, $e$ edges, and $p$ connected components is

$$v(G) = e - n + p(2)$$

In a strongly connected graph $G$, the cyclomatic number is equal to the maximum number of linearly independent circuits.

$V(G) = 9 - 6 + 2 = 5$ linearly independent circuits, e.g.,

- $(a \ b \ e \ f \ a)$
- $(b \ e \ b)$
- $(a \ b \ e \ a)$
- $(a \ c \ f \ a)$
- $(a \ d \ c \ f \ a)$

Suppose we view a program as a directed graph, an abstraction of its flow of control, and then measure the complexity by computing the number of linearly independent paths, $v(G)$.

Properties of Cyclomatic Complexity:

1) $v(G) \geq 1$
2) $v(G) = \#$ linearly independent paths 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 iff $v(G) = 1$
5) Inserting a new edge in $G$ increases $v(G)$ by 1
6) $v(G)$ depends only on the decision structure of $G$. 

McCabe
• For a collection of components
  – The cyclomatic number of the collection is the sum of the cyclomatic numbers of the individual components

\[ v(C) = \sum v(C_i), \text{ where } C = U C_i \]

• In the example: For more than 1 component

\[ v(M \cup A \cup B) = e - n + 2p = 13 - 13 + 2(3) = 6 \]

It can be shown that

\[ V(G) = \text{NUMBER OF DECISIONS} + 1 \]

The concept of cyclomatic complexity
  tied to to complexity of testing program
  easy to compute
  has been well studied

Cyclomatic complexity is used mostly on modules
  It has been recommended that the cyclomatic complexity of a module
  be kept small, e.g. less than 10

There is some evidence to support this
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?

How do these metrics compare with traditional size metrics such as the number of source lines of code or the number of executable statements?

How do these metrics relate to one another?

DEFINITIONS

EFFORT: The number of staff hours programmers and managers spend from the beginning of functional design to the end of acceptance testing.

QUALITY: The number of program faults reported during the development of the product.

Metric Evaluation in the SEL

Size and Complexity Measures Investigated

The Data:
Commercial software: Satellite Ground support software
Systems consist of 51K to 112K lines of FORTRAN source code
Ten to sixty one percent of the source code was modified from previous projects
Development effort ranges from 7K to 22K staff hours

This analysis focuses on:
Data from 7 projects
only newly developed modules (i.e., subroutines, functions, main procedures and block data)
Objective Size And Complexity Measures Investigated

Source Lines Of Code
Source Lines Of Code Excluding Comments
Executable Statements
Software Science Metrics
  N : Length In Operators And Operands
  V : Volume
  V* : Potential Volume
  L : Program Level
  E : Effort
  B : Bugs
Cyclomatic Complexity
Cyclomatic Complexity Excluding Compound Decisions
  (Referred To As Cyclo_cmplx_2)
Number Of Procedure And Function Calls
  Calls Plus Jumps
Revisions (Versions) Of The Source In The Program Library
Number Of Changes To The Source Code

Metric Evaluation in the SEL
Size and Complexity Measures Investigated

Spearman Correlations (R Values - All Signif. At \( P = 0.001 \))

<table>
<thead>
<tr>
<th></th>
<th>All Projects</th>
<th>Single Project</th>
<th>Single Programmer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validity Ratio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># Modules</td>
<td>731</td>
<td>79</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>31</td>
</tr>
<tr>
<td>E^^</td>
<td>.49</td>
<td>.70</td>
<td>.75</td>
</tr>
<tr>
<td>CYCLO_CMPLX_2</td>
<td>.47</td>
<td>.76</td>
<td>.79</td>
</tr>
<tr>
<td>CALLS &amp; JUMPS</td>
<td>.49</td>
<td>.78</td>
<td>.81</td>
</tr>
<tr>
<td>SOURCE LINES</td>
<td>.52</td>
<td>.69</td>
<td>.67</td>
</tr>
<tr>
<td>EXECUT. STMTS</td>
<td>.46</td>
<td>.69</td>
<td>.71</td>
</tr>
<tr>
<td></td>
<td>.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>.45</td>
<td>.68</td>
<td>.72</td>
</tr>
<tr>
<td></td>
<td>.80</td>
<td>.68</td>
<td></td>
</tr>
<tr>
<td>REVISIONS</td>
<td>.53</td>
<td>.68</td>
<td>.72</td>
</tr>
<tr>
<td></td>
<td>.68</td>
<td>.69</td>
<td></td>
</tr>
</tbody>
</table>

Some Relation To Effort Across All Projects

Relation Improve With:
  Individual Projects
  Validated Data
  Individual Programmers
Metric Evaluation in the SEL
Size and Complexity Measures Investigated

The Number Of Program Faults For A Given Module Is The Number Of System Changes That Listed The Module As Affected By An Error Correction

Weighted faults (w_flts) is A measure Of The amount Of effort spent isolating And fixing faults In A module

Spearman Correlation (R Values - All Signif. At P = 0.0001, Except (*) Signif. At P = 0.05)

<table>
<thead>
<tr>
<th>ALL PROJECTS</th>
<th>SINGLE PROJECT</th>
<th>SINGLE PROGRAMMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODULES</td>
<td>652</td>
<td>132</td>
</tr>
<tr>
<td>FAULTS</td>
<td>W_FLTS</td>
<td>FAULTS</td>
</tr>
<tr>
<td>E^2</td>
<td>.16</td>
<td>.19</td>
</tr>
<tr>
<td>CYCLO_CMPLX_2</td>
<td>.19</td>
<td>.20</td>
</tr>
<tr>
<td>CALLS &amp; JUMPS</td>
<td>.24</td>
<td>.25</td>
</tr>
<tr>
<td>SOURCE LINES</td>
<td>.26</td>
<td>.27</td>
</tr>
<tr>
<td>EXECUT. STMTS</td>
<td>.18</td>
<td>.20</td>
</tr>
<tr>
<td>B</td>
<td>.17</td>
<td>.19</td>
</tr>
<tr>
<td>REVISIONS</td>
<td>.38</td>
<td>.38</td>
</tr>
<tr>
<td>EFFORT</td>
<td>.32</td>
<td>.33</td>
</tr>
</tbody>
</table>

Relations Low Overall; # Revisions Strongest
Relations Improve With Individual Projects Or Programmers

Metric Evaluation in the SEL
Size and Complexity Measures Investigated

SPEARMAN R VALUES
(ALL SIGNIF. AT P = 0.001)

<table>
<thead>
<tr>
<th>SOURCE LINES (SLOC)</th>
<th>REV. SIONS</th>
<th>CALLS &amp; JUMPS</th>
<th>CALLS</th>
<th>CYCLO_CMPLX_2</th>
<th>CYCLO_CMPLX</th>
<th>EXECUT. STMTS</th>
<th>SLOC-CMMTS</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>E^2</td>
<td>.83</td>
<td>.37</td>
<td>.89</td>
<td>.62</td>
<td>.89</td>
<td>.88</td>
<td>.95</td>
<td>.86</td>
</tr>
<tr>
<td>V</td>
<td>.82</td>
<td>.35</td>
<td>.87</td>
<td>.57</td>
<td>.87</td>
<td>.87</td>
<td>.96</td>
<td>.86</td>
</tr>
<tr>
<td>SLOC-CMMTS</td>
<td>.93</td>
<td>.49</td>
<td>.88</td>
<td>.68</td>
<td>.86</td>
<td>.85</td>
<td>.91</td>
<td></td>
</tr>
<tr>
<td>EXECUT. STMTS</td>
<td>.85</td>
<td>.38</td>
<td>.91</td>
<td>.61</td>
<td>.92</td>
<td>.91</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CYCLO-CMPLX</td>
<td>.81</td>
<td>.39</td>
<td>.95</td>
<td>.55</td>
<td>.99</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CYCLO-CMPLX_2</td>
<td>.82</td>
<td>.38</td>
<td>.94</td>
<td>.56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALLS</td>
<td>.66</td>
<td>.41</td>
<td>.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALLS &amp; JUMPS</td>
<td>.85</td>
<td>.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REVISIONS</td>
<td>.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Metric Evaluation in the SEL
Size and Complexity Measures Investigated

Conclusions

Used commercially obtained data to validate software metrics
Common environment
Was able to perform validity checks and accuracy ratings

No metric
seemed to satisfactorily explain effort or development faults
related convincingly better with effort than the others

The strongest effort correlations
come from individual programmers or certain validated projects
increase with the more reliable data

The number of revisions correlates with development faults better than
any other metric

Many size and complexity measures relate well with each other

Product Models and Metrics

Data Structure Metrics

Data structure metrics measure the data interaction of the product

Major Concepts:

**Coupling:** refers to the degree of *interdependence between* parts of a
design, usually the amount of data shared by parts

**Cohesion:** refers to the degree of *dependence within* parts of the design,
usually the strength of the interaction

The definitions of these concepts depends on the design paradigm or
notation used to expressed the design. It is often programming
language dependent
Data Structure Metrics: Data bindings

- A SEGMENT - GLOBAL - SEGMENT DATA BINDING \((p, r, q)\) is an occurrence of the following:
  1. segment \(p\) modifies global variable \(r\)
  2. variable \(r\) is accessed by segment \(q\)
  3. \(p \neq q\)

- Existence of a data binding \((p, q, r)\) implies \(q\) dependent on the performance of \(p\) because of \(r\)

- DB \((p, r, q) \neq DB (q, r, p)\)

- \((p, r, q)\) represents a unique communication path between \(p\) and \(q\)

- The total \# Data Bindings represents the degree of a certain kind of "connectivity" (i.e., Between segment pairs via globals) within a complete program

---

INT  A, B, C, D

PROC P1
/* USES A, B */

PROC P2
/* USES A, B */

CALL P3 (X)

PROC P3 (INT E)
/* USES C, D */

PROC P4
/* USES C, D */

DATA BINDINGS
(P1, A, P2)
(P1, B, P2)
(P3, C, P4)
(P3, D, P4)
(P2, E, P3)

---

Basili/Turner
Data Structure Metrics: Data bindings

Levels of data binding (DB)

- **Potential DB** is an ordered triple \((p, x, q)\) for components \(p\) and \(q\) and variable \(x\) in the scope of \(p\) and \(q\).

- **Usage DB** is a potential DB such that \(p\) and \(q\) both use \(x\) for reference or assignment.

- **Feasible DB** (actual) is a usage DB such that \(p\) assigns to \(x\) and \(q\) references \(x\).

- **Control flow DB** is a feasible DB such that flow analysis allows the possibility of \(q\) being executed after \(p\) has started.

Data Structure Metrics

Segment Global Usage Pairs

- A segment-global usage pair \((p, r)\) is an instance of a global variable \(r\) being used by a segment \(p\) (i.e., \(r\) is either modified or set by \(p\)).

- Each usage pair represents a unique "use connection" between a global and a segment.

- Let actual usage pair (AUP) represent the count of true usage pairs (i.e., \(r\) is actually used by \(p\)).

- Let possible usage pair (PUP) represent the count of potential usage pairs (i.e., given the program's globals and their scopes, the scope of \(r\) contains \(p\) so that \(p\) could potentially modify \(r\)) (worst case).

- Then the relative percentage usage pairs (RUP) is \(RUP = \frac{AUP}{PUP}\) and is a way of normalizing the number of usage pairs relative to the problem structure.

- The RUP metric is an empirical estimate of the likelihood that an arbitrary segment uses an arbitrary global.
Measurement Across Time

Measures are sometimes difficult to understand in the absolute

However the relative changes in metrics over the evolution of the system can be very informative

This evolution may be within one development cycle of the product

\[ \text{e.g., Requirements} \rightarrow \text{Design} \rightarrow \text{Code} \rightarrow \ldots \]

Or

Multiple versions of the same product

\[ \text{e.g., Code}_1 \rightarrow \text{Code}_2 \rightarrow \text{Code}_3 \rightarrow \ldots \]

Measurement Across Time
Development/Maintenance Vector

A vector of metrics, \( m_1, m_2, m_n \), can be defined dealing with various aspects of the product, i.e., effort, changes, defects, logical, physical, and dynamic attributes, environmental considerations, ...

For example, some physical attributes might include

- (decisions, interaction of data, interaction of data, size)
- across modules within a module

The vector characterizes the product at some point in time

We can view it at various stages of product evolution to monitor how the product is changing

We can provide a set of bounds for the metrics to signal potential problems and anomalies
A vector of metrics, $m_1, m_2, m_n$, can be defined dealing with various aspects of the product, i.e., effort, changes, defects, logical, physical, and dynamic attributes, environmental considerations, ...

For example, some physical attributes might include (decisions, interaction of data, interaction of data, size) across modules within a module

The vector characterizes the product at some point in time

We can view it at various stages of product evolution to monitor how the product is changing

We can provide a set of bounds for the metrics to signal potential problems and anomalies

---

**Measurement Across Time**

**Development/Maintenance Vector**

---

Various metrics were used during different points in the development of a software product

The product was a compiler for a structured programming language
  – about 6,500 high level source statements
  – about 17,000 lines of source code

We will examine the changes of the values of various metrics across time
  – to provide insight into how the product was progressing
  – to allow us to evaluate the quality of the product

There were 17 enhancements of the product for this study
  – we will look at 5 major iterations
  – there were iterations after the last one

---

**Measurement Across Time**

**Case Study**

---
### Measurement Across Time
#### Case Study

**Statistics from Compilers at 5 Selected Points in the Iterative Process**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of statements</td>
<td>3404</td>
<td>4217</td>
<td>5181</td>
<td>5847</td>
<td>6350</td>
</tr>
<tr>
<td>Number of procedures</td>
<td>89</td>
<td>189</td>
<td>213</td>
<td>240</td>
<td>289</td>
</tr>
<tr>
<td>Number of separate Compiled modules</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>15</td>
<td>37</td>
</tr>
<tr>
<td>Average nesting level</td>
<td>3.4</td>
<td>2.9</td>
<td>2.9</td>
<td>2.9</td>
<td>2.8</td>
</tr>
<tr>
<td>Average number of Tokens per statement</td>
<td>5.7</td>
<td>6.3</td>
<td>6.6</td>
<td>7.2</td>
<td>7.3</td>
</tr>
</tbody>
</table>

### Measurement Across Time
#### Case Study

**Statistics from Compilers at 5 Selected Points in the Iterative Process**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage count of (Segment, Global) pairs (AUP)</td>
<td>611</td>
<td>786</td>
<td>941</td>
<td>1030</td>
<td>974</td>
</tr>
<tr>
<td>Total possible count of (Segment, Global) pairs</td>
<td>4128</td>
<td>8707</td>
<td>10975</td>
<td>6930</td>
<td>4584</td>
</tr>
<tr>
<td>Percentage use of global (PUP)</td>
<td>14.8</td>
<td>9.0</td>
<td>8.6</td>
<td>14.9</td>
<td>21.2</td>
</tr>
<tr>
<td>Total token size</td>
<td>19403</td>
<td>26567</td>
<td>34194</td>
<td>42098</td>
<td>46355</td>
</tr>
<tr>
<td>Number of Data Bindings</td>
<td>2610</td>
<td>6662</td>
<td>8759</td>
<td>12006</td>
<td>10442</td>
</tr>
<tr>
<td>Number of Data Bindings per Thousand tokens</td>
<td>13.4</td>
<td>17.7</td>
<td>25.6</td>
<td>28.5</td>
<td>22.5</td>
</tr>
</tbody>
</table>
Psychological Complexity: SPAN

• A SPAN is the number of statements between two textual references to the same identifier

\[
\begin{align*}
X &: = Y ; \\
Z &: = Y ; \\
X &: = Y ;
\end{align*}
\]

\[
\text{X - SPAN} \quad \text{Y - SPAN}
\]

• SPAN (X) = count of # statements between first and last statements (assuming no intervening references to X) Y has two spans

• For n appearances of an identifier in the source text, n - 1 spans are measured

• All appearances are counted except those in declare statements

• If SPAN > 100 statements, then one new item of information must be remembered for 100 statements till read again

Elshoff-GM

---

Psychological Complexity: SPAN

• COMPLEXITY \sim \# SPANS at any point (take max, average, median)

OR \sim \# Statements a variable must be remember (on the average) [average span]

VARIATION

• Do a live/dead variable analysis

• Complexity proportional to \# variables alive at any statement

• How does one scale up this measure?

\[
\begin{align*}
C (M) &= \# \text{STMTS} \\
&= \sum_{j=1}^{\# \text{stmts}} n_j \cdot s(n_j) \\
&\text{Where } n_j = \# \text{ spans of size } S(n_j)
\end{align*}
\]
Psychological Complexity: Variable SPAN

Variable span has been shown to be a reasonable measure of complexity

For commercial PL/1 programs, one study showed that a programmer must remember approximately 16 items of information when reading a program.

Product Models and Metrics

Dynamic Characteristics

We can divide the dynamic product the characteristics into two basic classes

We can view them as checking on the
  Behavior of the input to the code, e.g., coverage metrics
  Behavior of the code itself, e.g., reliability metrics
PRODUCT METRICS

Coverage Metrics

Based upon checking what aspects of the product are effected 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

TEST METRICS

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th># TEST CASES</th>
<th>SUBPROGRAM COVERAGE</th>
<th>BRANCH PATH COVERAGE</th>
<th>I/O COVERAGE</th>
<th>DO LOOP ENTRY</th>
<th>ASSIGNMENT</th>
<th>OTHER EXECUTABLE</th>
<th>CODE COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASCAL</td>
<td>32</td>
<td>.81</td>
<td>.39</td>
<td>.28</td>
<td>.92</td>
<td>.85</td>
<td>.74</td>
<td>.70</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>68</td>
<td>.91</td>
<td>1.00</td>
<td>.63</td>
<td>.92</td>
<td>.85</td>
<td>.74</td>
<td>.66</td>
</tr>
</tbody>
</table>
System engineering
   Determine the best trade-off between reliability and cost, schedule, etc.
   Optimize life cycle cost
   Specify reliability to the designer

Project management
   Progress monitoring
   Scheduling
   Investigation of alternatives

Operational software management

Evaluation of software engineering management

Musa, Goel, Littlewood

---

RELIABILITY
How Can We Use Reliability Metrics

Time dependent approaches
   Time between failures (Musa model)
   Failure counts in specified intervals (Goel/Okumoto)

Time-independent approaches
   Error seeding
   Input domain analysis

Problems with use of reliability models
   Lack of clear understanding of inherent strengths
   And weaknesses
   Underlying assumptions and outputs not fully understood by user
   Not all models applicable to all testing environments
RELIABILITY MODELS
Musa

Assumptions:
1. Errors are distributed randomly through the program
2. Testing is done with repeated random selection from the entire range of input data
3. The error discovery rate is proportional to the number of errors in the program
4. All failures are traced to the errors causing them and corrected before testing resumes
5. No new errors are introduced during debugging

\[ T = \frac{1}{KE} \times e^{kt} \]

WHERE
- \( E \) is total errors in the system
- \( t \) is the accumulated run time (starts @ 0)
- \( T \) is the mean time to failure

\[ m = M_0 \left[ 1 - \exp \left( \frac{-ct}{M_0 T_o} \right) \right] \]
RELIABILITY
Present MTTF vs Cumulative Execution Time

\[ T = T_0 e^{C_1/M_0 T_0} \]

INITIAL MTTF \( T_0 \)

PRESENT MTTF \( T \)

CUMULATIVE EXECUTION TIME

Musa

RELIABILITY
Software Reliability Estimation Execution Time Model

Musa
RELIABILITY
Combination of Approaches

Clean room
Developer uses reading techniques, top down development
Testing done by independent organization at incremental steps
Reliability model used to provide developer with quality assessment

Functional testing/coverage metrics
Use functional testing approach
Collect error distributions, e.g., Omission vs commission
Obtain coverage metrics
Knowing number of errors of omission, extrapolate

Error analysis and reliability models
Establish error history from previous projects
Distinguish similarities and differences to current project
Determine prior error distributions for the current project
Select a class of stochastic models for the current project
Update prior distributions and compare actual data with the priors for the current project

Automatable Change And Error Metrics

Automatable Metric
No interference to the developer
Computed algorithmically from quantifiable sources
Reproducible on other projects with the same algorithms

Useful Metric
Sensitive to externally observable differences in the development environment
Relative values correspond to some intuitive notion about characteristic differences in the environment

Examples
Program changes: Textual revision in the source code representing one conceptual change
Job steps: The number of computer accesses during development or maintenance
Automatable Change And Error Metrics

Program Changes

Textual revisions in the source code of a module during the development period
One program change should represent one conceptual change to the program

A program change is defined as:
One or more changes to a single statement
One or more statements inserted between existing statements
A change to a single statement followed by the insertion of new statements

The following are not counted as program changes:
The deletion of one or more existing statements
The insertion of standard output statements or special compiler-provided debugging directives
The insertion of blank lines or comments, the revision of comments and reformatting without alteration of existing statements

Program changes have been shown to correlate well with faults

Automatable Change And Error Metrics

Job Steps

The number of computer accesses

A single programmer-oriented activity performed on the computer at the operating system command level

Basic to the development effort and involves non-trivial expenditures of computer or human resources

Examples: text editing, module compilation, program compilation, link editing, program execution
Product Models and Metrics

References


