

Change Models and Metrics

Change and Defect 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., effects certain customer classes

Change and Defect Models and Metrics

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

Change and Defect Models and Metrics

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

Faults

Concrete manifestations of errors within the software

- One error may cause several faults
- Various errors may cause identical faults

Failures

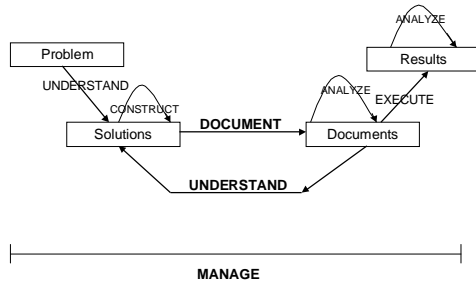
Departures of the operational software system behavior from user expected requirements

- A particular failure may be caused by several faults
- Some faults may never cause a failure (difference between reliability and correctness)

IEEE/Standard

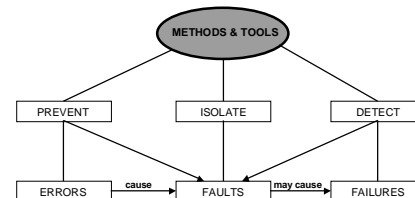
Building Models

Methods and Tools Dealing with Software Defects



Building Models

Methods and Tools Dealing with Defects



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 performers conformance to the process definition
process performers understanding of the domain
to which the process is being applied

Process metrics can be used to
evaluate the process
gain insight into the product
find weakness in the environment in which the process is being applied
provide insight into process improvement
help tailor and evolve the processes over time

PROCESS METRICS Building Process Models

Modeling the education and training process

- Process:**
1. provide the individual with training manuals they must read
 2. provide a course, educating the individual in the process
 3. provide training by applying the process to a toy problem
 4. assign the individual to a project that is using the process, mentored by an experienced method user
 5. after this the individual is considered fully trained in the process

Convert to 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 had been 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

Product Data

We will divide the static product the metrics into three basic classes
Size metrics
Control Structure Metrics
Data Structure Metrics

Size metrics attempt to measure the physical size of the product

Control structure metrics attempt to measure the control flow of the product

Data structure metrics attempt to 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.

Dynamic metrics can be viewed as checking on the
Behavior of the input to the code, e.g., coverage metrics
Behavior of the code, e.g., reliability metrics

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

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

PRODUCT METRICS

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

PRODUCT 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

| | SIMPLE | AVERAGE | COMPLEX |
|--------------------|--------|---------|---------|
| Input | 3 | 4 | 6 |
| Output type | 4 | 5 | 7 |
| Query type | | | |
| - Input part | 3 | 4 | 6 |
| - Output part | 4 | 5 | 7 |
| File type | 7 | 10 | 15 |
| External interface | 5 | 7 | 10 |

Function Points Calculation

Application Characteristics

| | |
|--------------------------------|--------------------------------|
| DATA COMMUNICATIONS | ON-LINE UPDATE |
| DISTRIBUTED DATA OR PROCESSING | COMPLEX PROCESSING |
| PERFORMANCE OBJECTIVES | REUSABILITY |
| HEAVILY-USED CONFIGURATION | CONVERSION & INSTALLATION EASE |
| TRANSACTION RATE | OPERATIONAL EASE |
| ON-LINE DATA ENTRY | MULTIPLE-SITE |
| END USER EFFICIENCY | FACILITATE CHANGE |

INFLUENCE SCALE

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

Function Points Calculation

FUNCTION POINTS =

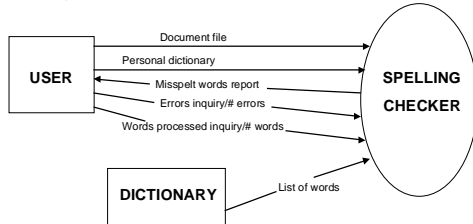
$$(\sum \text{INPUTS} * \text{WEIGHTS} + _ \text{OUTPUTS} * \text{WEIGHTS})$$

$$+ \sum \text{QUERIES} * \text{WEIGHTS} + _ \text{FILES} * \text{WEIGHTS} +$$

$$+ \sum \text{INTERFACES} * \text{WEIGHTS}) * (0.65 + 1\% \text{ TOTAL INFLUENCE})$$

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.



Function Points Calculation Example

- INPUTS: DOCUMENT FILE NAME, PERSONAL DICTIONARY NAME
- OUTPUT: MISSPELT WORDS REPORT, # WORDS PROCESSED MESSAGE, # ERRORS MESSAGE
- QUERIES: ERRORS FOUND, WORDS PROCESSED
- FILES: DICTIONARY
- INTERFACES: DOCUMENT FILE, PERSONAL DICTIONARY

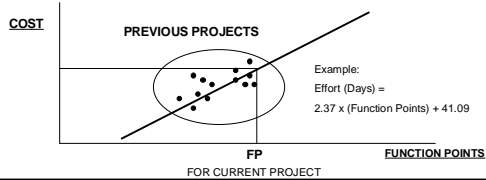
ASSUMING AVERAGE COMPLEXITY IN EACH CASE AND MINOR IMPACT

| | | | |
|------------|---|----|----|
| INPUTS | 2 | 4 | 8 |
| OUTPUTS | 3 | 5 | 15 |
| QUERIES | 2 | 9 | 18 |
| FILES | 1 | 10 | 10 |
| INTERFACES | 2 | 7 | 14 |

$$65 \times (0.65 + 0.01) = 51.35 \text{ FUNCTION POINTS}$$

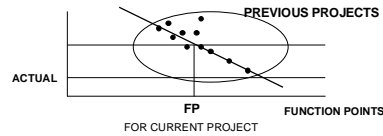
Function Points Estimating Costs

- COST ESTIMATION USING FUNCTION POINTS REQUIRES
 - COST DATA FOR PREVIOUS PROJECTS
 - FUNCTION POINTS COUNTED IN PREVIOUS PROJECTS
- THE ESTIMATION PROCESS:
 - THE DATA ABOUT THE COST 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



Function Points Assessing Productivity

- PRODUCTIVITY ASSESSMENT USING FUNCTION POINTS REQUIRES
 - PRODUCTIVITY FIGURES FOR PREVIOUS PROJECTS
 - FUNCTION POINTS COUNTED IN PREVIOUS PROJECTS
- THE ASSESSMENT PROCESS:
 - THE DATA ABOUT THE PRODUCTIVITY IN PREVIOUS PROJECTS IS PLOTTED AGAINST THE FUNCTION POINTS COUNTED IN THOSE PROJECTS
 - THE EXPECTED PRODUCTIVITY IS THE PRODUCTIVITY VALUE FOR THE FUNCTION POINTS OF THIS PROJECT
 - DISCREPANCIES BETWEEN ACTUAL 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 SATISFY THE CRITERION
- EXAMPLE: A RECENT STUDY CONDUCTED IN A CANADIAN FINANCIAL INSTITUTION ON MAINTENANCE ACTIVITIES (21 PROJECTS, 332 AVERAGE STAFF DAYS PER PROJECT, MIN 52, MAX 531)

| DEVIATION | PROJECTS WITHIN RANGE | |
|-----------|-----------------------|-----|
| | NUMBER | % |
| +/- 10% | 9 | 43% |
| +/- 20% | 12 | 57% |
| +/- 26% | 17 | 81% |

Software Science

- MEASURABLE PROPERTIES OF ALGORITHMS

- n_1 = # Unique or distinct operators in an implementation
- n_2 = # Unique or distinct operands in an implementation
- N_1 = # Total usage of all operators
- N_2 = # Total usage of all operands
- $f_{1,j}$ = # Occurrences of the j^{th} most frequent operator $j = 1, 2, \dots, n_1$
- $f_{2,j}$ = # Occurrences of the j^{th} most frequent operand $j = 1, 2, \dots, n_2$

THE VOCABULARY n IS

$$n = n_1 + n_2$$

THE IMPLEMENTATION LENGTH IS

$$N = N_1 + N_2$$

$$\text{and } N_1 = \sum_{j=1}^{n_1} f_{1,j} \quad N_2 = \sum_{j=1}^{n_2} f_{2,j} \quad N = \sum_{j=1}^{n_1} f_{1,j} + \sum_{j=1}^{n_2} f_{2,j}$$

Example: Euclid's Algorithm

```

IF (A = 0)
LAST:   BEGIN GCD := B; RETURN END;
        IF (B = 0)
        BEGIN GCD := A; RETURN END;

HERE:   G := A/B; R := A - B X G;
        IF (R = 0) GO TO LAST;
        A := B; B := R; GO TO HERE
    
```

Operator Parameters Greatest Common Divisor Algorithm

| OPERATOR | j | f _{1j} |
|-------------------|---------------------|---------------------|
| ; | 1 | 9 |
| := | 2 | 6 |
| () or BEGIN...END | 3 | 5 |
| IF | 4 | 3 |
| = | 5 | 3 |
| / | 6 | 1 |
| - | 7 | 1 |
| x | 8 | 1 |
| GO TO HERE | 9 | 1 |
| GO TO LAST | 10 | 1 |
| | n ₁ = 10 | N ₁ = 31 |

Operand Parameters Greatest Common Divisor Algorithm

| OPERAND | j | f _{2j} |
|---------|--------------------|---------------------|
| B | 1 | 6 |
| A | 2 | 5 |
| O | 3 | 3 |
| R | 4 | 3 |
| G | 5 | 2 |
| GCD | 6 | 2 |
| | n ₂ = 6 | N ₂ = 21 |

Software Science Metrics

PROGRAM LENGTH:

$$n - \hat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

\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)

$$V = N \log_2 n$$

B = The number of bits necessary to represent the program

Software Science Metrics

POTENTIAL VOLUME:

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

Where n₂ represents the number of input/output parameters

V = A measure of the specification for an algorithm

PROGRAM LEVEL: (Level of an implementation)

$$L = V / V$$

$$L = \frac{2n_2}{n_1 N_2} \sim L$$

D' = 1/L = Difficulty

Software Science Majors

PROGRAMMING EFFORT:

$$E = V D = V/L = V^2/N^*$$

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

E = A measure of program clarity

TIME:

$$T = E/S = V/SL = V^2/SV^*$$

T = the time to develop an algorithm

ESTIMATED BUGS:

$$\hat{B} = (LE)/E_0 = V/E_0$$

WHERE E₀ = The mean effort between potential errors in programming

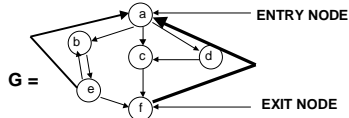
\hat{B} = the number of errors expected in a program

Cyclomatic Complexity

- THE **CYCLOMATIC NUMBER** $V(G)$ OF A GRAPH G WITH n VERTICES, e EDGES, AND p CONNECTED COMPONENTS IS

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

- 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)

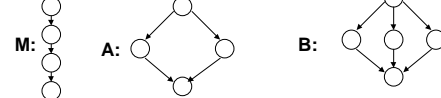
- OVERALL STRATEGY IS TO MEASURE THE COMPLEXITY OF A PROGRAM BY COMPUTING THE NUMBER OF LINEARLY INDEPENDENT PATHS $V(G)$

McCabe

Properties of Cyclomatic Complexity

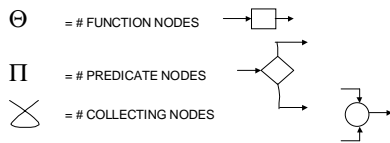
- $V(G) \geq 1$
- $V(G) = \#$ LINEARLY INDEPENDENT PATHS IN G ; IT IS THE SIZE OF A BASIS SET
- INSERTING OR DELETING FUNCTIONAL STATEMENTS TO G DOES NOT AFFECT $V(G)$
- G HAS ONLY ONE PATH $\implies V(G) = 1$
- INSERTING A NEW EDGE IN G INCREASES $V(G)$ BY 1
- $V(G)$ DEPENDS ONLY ON THE DECISION STRUCTURE 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 $V(C) = \sum_i V(C_i)$ $C = \cup C_i$

Simplification



THEN

$$e = 1 + \theta + 3\pi$$

$$n = \theta + 2\pi + 2$$

ASSUMING $p = 1$ AND $v = e - n + 2p$ YIELDS
 $v = (1 + \theta + 3\pi) - (\theta + 2\pi + 2) + 2 = \pi + 1$

Simplification

THE RESULT GENERALIZES TO NONSTRUCTURED PROGRAMS

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

CONCEPT IS TIED TO TESTABILITY
 INTUITIVELY SATISFYING METRIC FOR COMPLEXITY
 EASY TO COMPUTE
 WELL STUDIED

McCABE RECOMMENDS A MAXIMUM $V(G)$ OF 10 FOR ANY MODULE

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 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

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'S)

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