

# Product Models and Metrics

## Product Models and Metrics

---

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

or

- a nominal class, e.g., the component represents a mathematical  
function, data structure, ...

3

## 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.

4

## 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

5

## 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

6

## PRODUCT METRICS

---

### Documentation Metrics

**(Law of the Core Dump)** The thickness of the proposal required to win a multimillion dollar contract is about one millimeter per million dollars. If all the proposals conforming to this standard were piled on top of each other at the bottom of the Grand Canyon, it would probably be a good idea.

-Norman Augustine (Former CEO,  
Lockheed Martin Corp.)

7

## 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 8

## 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

9

## 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)

10

## Function Points Calculation Complexity Weights

	SIMPLE	AVERAGE	COMPLEX
<b>Input</b>	3	4	6
<b>Output type</b>	4	5	7
<b>Query type</b>			
— Input part	3	4	6
— Output part	4	5	7
<b>File type</b>	7	10	15
<b>External interface</b>	5	7	10

11

## Function Points Calculation

### Application Characteristics

DATA COMMUNICATIONS  
 DISTRIBUTED DATA OR PROCESSING  
 PERFORMANCE OBJECTIVES  
 HEAVILY-USED CONFIGURATION  
 TRANSACTION RATE  
 ON-LINE DATA ENTRY  
 END USER EFFICIENCY

ON-LINE UPDATE  
 COMPLEX PROCESSING  
 REUSABILITY  
 CONVERSION & INSTALLATION EASE  
 OPERATIONAL EASE  
 MULTIPLE-SITE  
 FACILITATE CHANGE



INFLUENCE SCALE

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

12

## Function Points Calculation

---

FUNCTION POINTS =

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

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

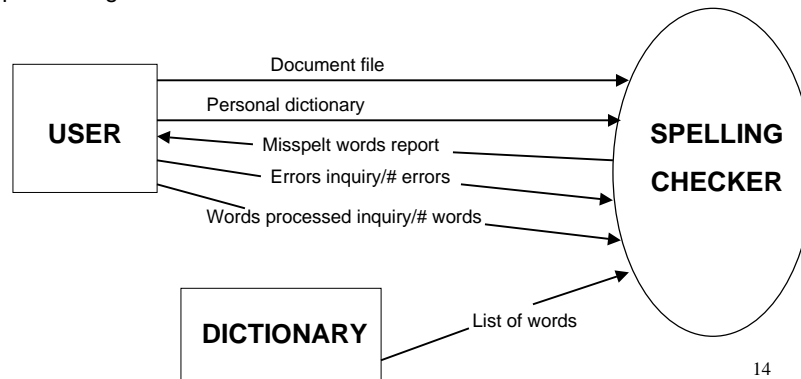
$\Sigma \text{ INTERFACES} * \text{WEIGHTS}) * (0.65 + 1\% \text{ TOTAL INFLUENCE})$

13

## 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.



14

## 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

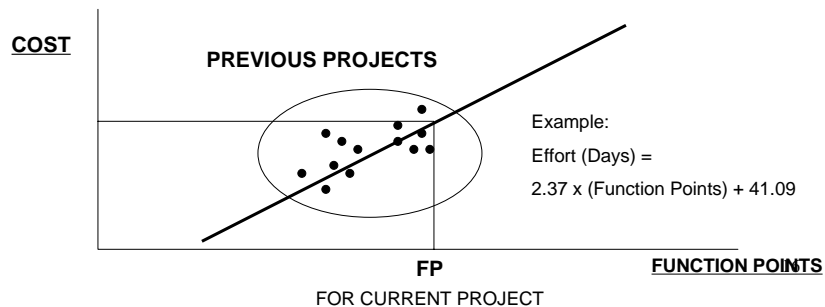
$$65 * (0.65 + 0.01) = 51.35 \text{ FUNCTION POINTS}$$

15

## Function Points Estimating Costs

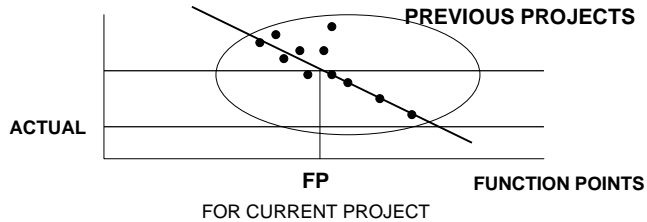
---

- 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



17

## 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%

18

## 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 in the early 1970s. It is based upon algorithmic complexity theory concepts previously developed by Chaitin and Kolmogorov in the 1960s, as extensions to Shannon's work on information theory.

19

## Algorithmic complexity

---

- The randomness (or algorithmic complexity) of a string is the minimal length of a program which can compute that string.
- Considered the 192 digits in the string 123456789101112...9899100.
  - Its complexity is less than 192 since it can be described by the 27 character Pascal statement for l:=1 to 100 do write(l).
  - But a random string of 192 digits would have no shorter encoding. The 192 characters in the string would be its own encoding.
  - Therefore the given 192 digits is less random and more structured than 192 random digits.
- If we increase the string to 5,888,896 digits 1234...9999991000000, then we only need to marginally increase the program complexity from 27 to 31 for l:=1 to 1000000 do write(l).
  - The 5,888,704 additional digits only add 4 characters of complexity to the string.
  - It is by no means that 31 is even minimal. It is assuming a Pascal interpreter. The goal of Chaitin's research was to find the absolute minimal value for any sequence of data items.
- Halstead tried to apply these concepts to estimating program size.

20

## 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  $j^{\text{th}}$  most frequent operator  
 $j = 1, 2, \dots, n_1$   
 $f_{2,j}$  = # Occurrences of the  $j^{\text{th}}$  most frequent operand  
 $j = 1, 2, \dots, n_2$

THE VOCABULARY  $n$  IS

$$n = n_1 + n_2$$

THE IMPLEMENTATION LENGTH IS

$$N = N_1 + N_2$$

and

$$N_1 = \sum_{j=1}^{n_1} f_{1,j} \quad N_2 = \sum_{j=1}^{n_2} f_{2,j} \quad N = \sum_{i=1}^2 \sum_{j=1}^{n_i} f_{ij}$$

21

## Example: Euclid's Algorithm

---

LAST:           IF (A = 0)  
                   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

22

**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
	$n_1 = 10$	$N_1 = 31$

23

**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
	$n_2 = 6$	$N_2 = 21$

24

## Software Science Metrics

---

PROGRAM LENGTH:

$$n \sim N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

$N$  = 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

25

## Software Science Metrics

---

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)

$$L = V / V$$

$$L = 2n_2^{-L}$$

$$D = \frac{1}{L} = \text{Difficulty}$$

26

## Software Science Metrics

---

### PROGRAMMING EFFORT:

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

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  $E_0$  = The mean effort between potential errors in programming

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

27

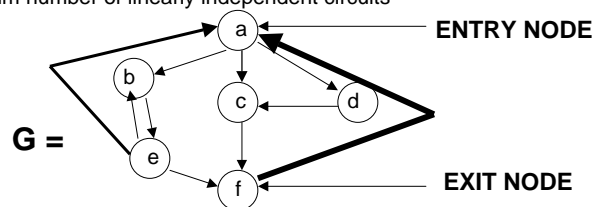
## Cyclomatic Complexity

---

- 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)

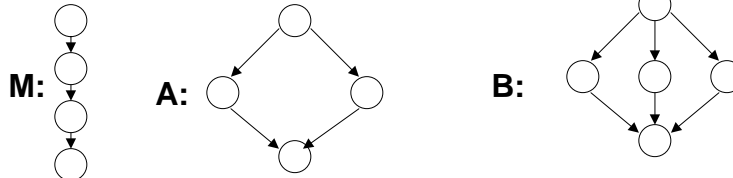
- Suppose we view a program as a directed graph, an abstraction of its flow of control, and then measure the complexity by computing the number of linearly independent paths,  $v(G)$

28  
McCabe

## 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  $G$

- For more than 1 component



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

- For a collection of components  
 $v(C) = \sum_v (C_i)$                        $C = \cup C_i$

29

## Simplification

$\Theta$  = # Function nodes

$\Pi$  = # Predicate nodes

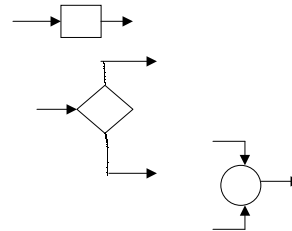
= # Collecting nodes



THEN

$$e = 1 + \Theta + 3\Pi$$

$$n = \Theta + 2\Pi + 2$$



ASSUMING  $p = 1$  AND  $v = e - n + 2p$  YIELDS

$$v = (1 + \Theta + 3\Pi) - (\Theta + 2\Pi + 2) + 2 = \Pi + 1$$

I.E.,  $v(G)$  of a structured program equals the number of predicate nodes plus 1

30

## **Simplification**

---

THE RESULT GENERALIZES TO NONSTRUCTURED PROGRAMS

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

The concept of cyclomatic complexity is tied to to complexity of testing program

As a metric, it is easy to computer

It has been well studied

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

31

## **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. 32

**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<sup>3</sup>  
DATA'S)

**METRIC EVALUATION IN THE SEL**  
**SIZE and COMPLEXITY MEASURES INVESTIGATED**

**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\_CMLPX\_2)

NUMBER OF PROCEDURE AND FUNCTION CALLS

CALLS PLUS JUMPS

REVISIONS (VERSIONS) OF THE SOURCE IN THE PROGRAM LIBRARY 34

NUMBER OF CHANGES TO THE SOURCE CODE

## METRICS' RELATION TO ACTUAL EFFORT

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

	ALL PROJECTS		SINGLE PROJECT		SINGLE PROGRAMMER
	ALL	ALL	80%	90%	92.5%
VALIDITY RATIO					
# MODULES	731	79	29	20	31
E^^	.49	.70	.75	.80	.79
CYCLO_CMPLX_2	.47	.76	.79	.79	.68
CALLS & JUMPS	.49	.78	.81	.82	.70
SOURCE LINES	.52	.69	.67	.73	.86
EXECUT. STMTS	.46	.69	.71	.78	.75
V	.45	.68	.72	.80	.68
REVISIONS	.53	.68	.72	.80	.68

SOME RELATION TO EFFORT ACROSS ALL PROJECTS

RELATION IMPROVE WITH:

INDIVIDUAL PROJECTS

VALIDATED DATA

INDIVIDUAL PROGRAMMERS

35

## METRICS' RELATION TO PROGRAM FAULTS

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)

	ALL PROJECTS		SINGLE PROJECT		SINGLE PROGRAMMER	
	652	652	132	132	21	21
MODULES	FAULTS	W_FLTS	FAULTS	W_FLTS	FAULTS	W_FLTS
E^^	.16	.19	.58	.52	.67	.65
CYCLO_CMPLX_2	.19	.20	.55	.49	.48*	.45*
CALLS & JUMPS	.24	.25	.57	.52	.60*	.56*
SOURCE LINES	.26	.27	.65	.62	.66	.65
EXECUT. STMTS	.18	.20	.54	.51	.58*	.53*
B	.17	.19	.54	.50	.68	.66
REVISIONS	.38	.38	.78	.69	.83	.81
EFFORT	.32	.33	.64	.62	.67	.62

RELATIONS LOW OVERALL; # REVISIONS STRONGEST  
RELATIONS IMPROVE WITH INDIVIDUAL PROJECTS OR PROGRAMMERS

36

## RELATIONSHIP BETWEEN SIZE and COMPLEXITY METRICS

### SPEARMAN R VALUES

(ALL SIGNIF. AT P = 0.001)

1794 MODULES

	SOURCE LINES (SLOC)	REVI- SIONS	CALLS & JUMPS	CALLS	CYCLO- CMLX_	CYCLO- CMLX	EXECUT STMTS	SLOC- CMMTS	V
E^	.83	.37	.89	.62	.89	.88	.95	.86	.98
V	.82	.35	.87	.57	.87	.87	.96	.86	
SLOC- CMMTS	.93	.49	.88	.68	.86	.85	.91		
EXECUT STMTS	.85	.38	.91	.61	.92	.91			
CYCLO- CMLX	.81	.39	.95	.55	.99				
CYCLO- CMLX_2	.82	.38	.94	.56					
CALLS	.66	.41	.75						
CALLS & JUMPS	.85	.44							
REVI- SIONS	.50								

37

## CONCLUSIONS

**CAN** USE COMMERCIALY OBTAINED DATA TO VALIDATE SOFTWARE METRICS

**VALIDITY** CHECKS AND ACCURACY RATINGS ARE USEFUL

**NONE** OF THE METRICS SEEM TO SATISFACTORILY EXPLAIN EFFORT OR DEVELOPMENT FAULTS

**NEITHER** SOFTWARE SCIENCE'S **E** METRIC, CYCLOMATIC COMPLEXITY NOR SOURCE LINES RELATES CONVINCINGLY BETTER WITH EFFORT THAN THE OTHERS

**THE STRONGEST** EFFORT CORRELATIONS ARE DERIVED WITH MODULES FROM INDIVIDUAL PROGRAMMERS OR CERTAIN VALIDATED PROJECTS

**THE MAJORITY** OF EFFORT CORRELATIONS INCREASE WITH THE MORE RELIABLE DATA

THE NUMBER OF REVISIONS CORRELATES WITH DEVELOPMENT FAULTS BETTER THAN EITHER SOFTWARE SCIENCE'S **B** METRIC, **E** METRIC, CYCLOMATIC COMPLEXITY OR SOURCE LINES OF CODE

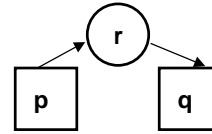
SOME OF THE SIZE AND COMPLEXITY MEASURES RELATE WELL WITH EACH OTHER

38

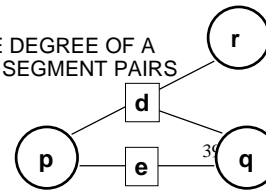
## DATA BINDINGS

- 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 ≠ q



- EXISTENCE OF A DATA BINDING (p, r, q) ⇔ q DEPENDENT ON THE PERFORMANCE OF p BECAUSE OF r
- BINDING (p, r, q) ≠ BINDING (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 "CONNECTIVITY" (I. E., BETWEEN SEGMENT PAIRS VIA GLOBALS) WITHIN A COMPLETE PROGRAM



**BASIL/TURNER**

## INT A, B, C, D

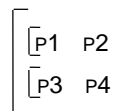
```

PROC P1
  /* USES A, B */
  . . . .
PROP 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)



40

## 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

BASIL/HUTCHENS

41

## 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
$$\mathbf{RUP = AUP/PUP}$$
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

42

## METRICS ACROSS TIME Measurement Across Time

Measures are sometimes difficult to understand in the absolute

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

This evolution may be within one development cycle of the product

e.g., **Requirements --> Design --> Code --> ...**

Or

Multiple versions of the same product

e.g., **Code<sub>1</sub> --> Code<sub>2</sub> --> Code<sub>3</sub> --> ...**

43

## METRICS ACCROSS 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

44

## METRICS ACROSS TIME Case Study

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

45

## METRICS ACROSS TIME Statistics from Compilers at 5 Selected Points in the Iterative Process

	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>
• NUMBER OF STATEMENTS	3404	4217	5181	5847	6350
• NUMBER OF PROCEDURES AND FUNCTIONS	89	189	213	240	289
• NUMBER OF SEPARATE COMPILED MODULES	4	4	7	15	37
• AVERAGE NESTING LEVEL	3.4	2.9	2.9	2.9	2.8
• AVERAGE NUMBER OF TOKENS PER STATEMENT	5.7	6.3	6.6	7.2	7.3

46

## METRICS ACROSS TIME

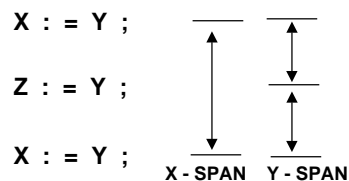
### Statistics from Compilers at 5 Selected Points in the Iterative Process

	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>
USAGE COUNT OF (SEGMENT, GLOBAL) PAIRS	611	786	941	1030	974
TOTAL POSSIBLE COUNT OF (SEGMENT, GLOBAL) PAIRS	4128	8707	10975	6930	4584
PERCENTAGE USE OF GLOBALS	14.8	9.0	8.6	14.9	21.2
NUMBER OF ACTUAL DATA BINDINGS	2610	6662	8759	12006	10442
NUMBER OF POSSIBLE DATA BINDINGS	243780	814950	1337692	497339	342727
PERCENTAGE OF POSSIBLE BINDINGS THAT OCCURED	1.1	0.8	0.7	2.4	3.0

47

## SPAN

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



- $SPAN(X) = \text{count of \# statements between first and last statements (assuming no intervening references to X)}$  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

48

(CON'T)

**Elshoff-GM**

## SPAN

- COMPLEXITY ~ # SPANS at any point (take max, average, median)  
OR ~ # 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?

$$C(M) = \frac{\sum_{j=i}^{\# \text{ STMTS}} n_j \cdot s(n_j)}{\# \text{ stmts}}$$

Where  $n_i = \#$  spans of size  $S(n_i)$

49

## 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

Elshoff-GM

50

## 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

51

## 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 if the requirements documents have been read

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

52

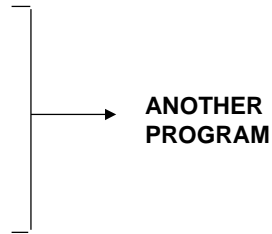
## TEST METRICS

### PASCAL

<b># TEST CASES:</b>	<b>32</b>	<b>36</b>
SUBPROGRAM COVERAGE	.81	.92
BRANCH PATH COVERAGE	.59	.67
I/O COVERAGE	.23	.54
DO LOOP ENTRY	.92	.94
ASSIGNMENT	.85	.91
OTHER EXECUTABLE	.74	.78
CODE COVERAGE	.70	.80

### FORTRAN

<b># TEST CASES:</b>	<b>68</b>
SUBROUTINE COVERAGE	.91
FUNCTION COVERAGE	1.00
BRANCH PATH	.63
I/O	.35
DO-LOOP	.74
ASSIGNMENT	.48
OTHER EXECUTABLE	.66



53  
STUCKI - BOEING

## RELIABILITY How Can We Use Reliability Metrics

### 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

54

## RELIABILITY Software Reliability Models

### 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 55

## 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

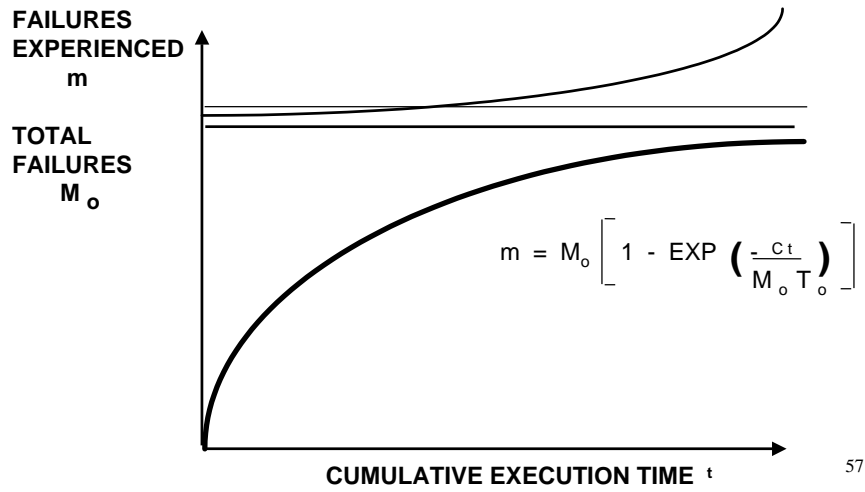
$$T = \frac{1}{KE} e^{Kt}$$

WHERE **E** IS TOTAL ERRORS IN THE SYSTEM  
**t** IS THE ACCUMULATED RUN TIME (STARTS @ 0)  
**T** IS THE MEAN TIME TO FAILURE

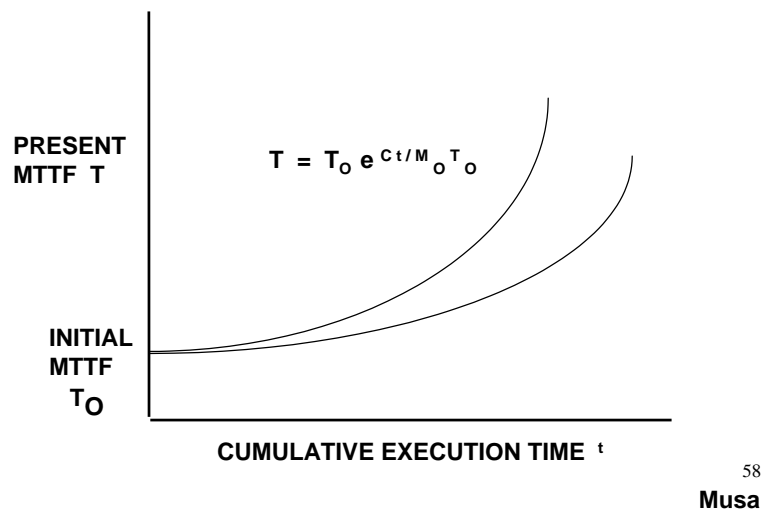
56

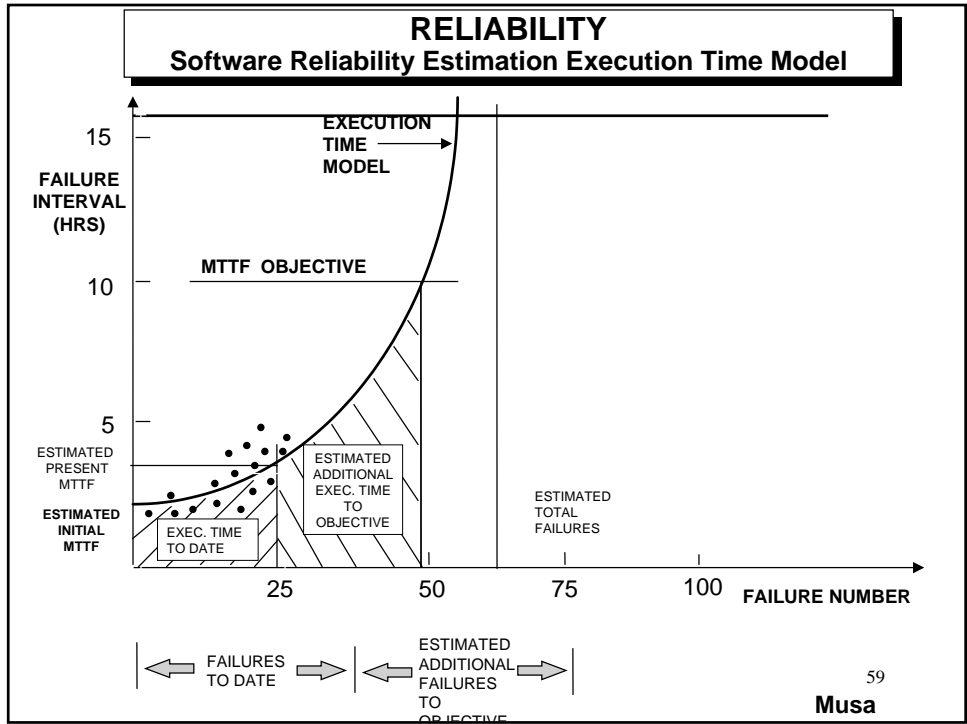
**Musa, Iannino, Okumoto - Software Reliability**

### RELIABILITY Failures Experienced vs Cumulative Execution Time



### RELIABILITY Present MTTF vs Cumulative Execution Time





### 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

60

## **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

61

## **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

62

GANNON / DUNSMORE

## **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**

**Basili / Reiter**

63

## **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

64

## 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

65

### Additional Augustine Laws Related to Metrics

---

- One tenth of the participants produce over one third of the output. Increasing the number of participants merely reduces the average output. (A variation on Brook's Law-Adding people to speed up a late project just makes it later. )
- The last 10% of performance generates one third of the cost and two thirds of the problems.
- (Law of Unmitigated Optimism) Any task can be completed in only one-third more time than is currently estimated.
- (Law of Inconstancy of Time) A revised schedule is to business what a new season is to an athlete or a new canvas to an artist.
- Law of Propagation of Misery) If a sufficient number of management layers are superimposed on top of each other, it can be assured that disaster is not left to chance.
- VP of Boeing on 767 project: "is further ahead at the halfway point than any new airliner program in Boeing history."

66