STUDIES OF DEFECTS

DEFECT CHARACTERIZATION

Goal

Analyze the **DOS/VS Operating system release 28** in order to **characterize** it with respect to the **defects, interface defects, types of misunderstanding** from the point of view of the **organization**.

Environment:

- IBM Germany
- Operating System Release
  - ~ 500 modules affected by the modification
  - average size ~360 LOC (480LOC with comments)
- 432 faults reported

Experimental design:

- Single project/case study
- in vivo, no report on experience

Endres

DEFECT CHARACTERIZATION

Definitions

Defect:

Number of defect report forms

Interface defect:

more than one module affected by defect fix

Misunderstanding type:

Problem specific, implementation specific, textual specific

DEFECT CHARACTERIZATION

Defect Distribution by Modules

Number of Defects		Number of Modules Affected
371	(85%)	1
50		2
6		3
3		4
1		5
1		8

DEFECT CHARACTERIZATION

Defect Distribution by Modules

Number of Defects		Number of Modules Affected
371	(85%)	1
50		2
6		3
3		4
1		5
1		8

DEFECT CHARACTERIZATION

Defect Distribution by Modules

Number of Defects per Module

Number of Modules	Number of Defects/Modules
112	1
36	2
15	3
11	4
8	5
2	6
4	7
5	8
3	9
2	10
1	14
1	15
1	19
1	28

DEFECT CHARACTERIZATION

Problem Specific Defects

Machine Configuration and Architecture	10
Dynamic Behavior and Communication between Processes	17
Functions Offered	12
Output Listings and Formats	3
Diagnostics	3
Performance	1
	46%

DEFECT CHARACTERIZATION

Implementation Specific Defects

Initialization (of Fields and Areas)	8
Addressability (in the sense of the assembler)	7
Reference to Names	7
Counting and Calculating	8
Masks and Comparisons	2
Estimation of Range Limits (addresses and parameters)	1
Placing of instructions within a module (bad fixes)	5
	38%

DEFECT CHARACTERIZATION

Textual Defects

Spelling in messages and commentaries	4
Missing commentaries or flowcharts (standards)	5
Incompatible status of macros or modules (integration defects)	5
Not classifiable	2
	16%

DEFECT CHARACTERIZATION

Conclusions

Interface between modules not a major source of defects

Approximately half the defects originated in misunderstanding of the problem to be solved or potential solutions

(=> not susceptible to improved programming techniques,

I.e., a higher order programming language)

Consider this as

Analyze the **defects** in order to **characterize** them with respect to the **various classification schemes** from the point of view of the **knowledge builder**.in the context of

a single version of an operating system, ...

REQUIREMENTS DOCUMENT EVALUATION

Goal

Analyze the **SCR requirements method** in order to **evaluate** it with respect to the **ease of modification and quality of the requirements document produced** from the point of view of the **organization**.

Environment:

Naval Research Laboratory
On-board flight program for the A-7 aircraft
real-time, interactive, using TC-2
computer 16K 16 bit words
Data collected after document baselined

Experimental design:

Single project/case study
in vivo, experience in method, novices in application

Basili/Weiss

REQUIREMENTS DOCUMENT EVALUATION

Goal

Analyze the **method** in order to **evaluate** it with respect to the **effect on product** from the point of view of the **organization**.

Analyze the **SCR method** in order to **evaluate** it with respect to the
ease of modification of the requirements document
structuredness of the requirements document
ability to minimize consistency and ambiguity faults
from the point of view of the **organization**.

Analyze the **requirements document** in order to **characterize** it with respect to the **ease of modification, structure, and consistency and un-ambiguousness** from the point of view of **quality assurance**.

Analyze the **use of the requirements document** in order to **characterize** it with respect to the **its worthiness to be maintained** from the point of view of the **organization**.

REQUIREMENTS DOCUMENT EVALUATION

Process Questions

Process conformance:

What is the requirements development methodology?

(formal specifications using a state machine model)

How well is it being applied?

(developers were experimenting with the methodology)

Domain understanding:

How well do the developers understand the application?

(they had minimal expertise)

REQUIREMENTS DOCUMENT EVALUATION

Product Questions

Product dimensions:

What is the size of the requirements document? (462 pages)

Cost:

What is the staff effort expended in producing the document?
(17 staff months)

What is the staff effort expended in making the changes?
(11 staff weeks)

What is the total staff effort expended in development during the time the data was collected? (122 staff weeks)

What is the calendar time for development during the time the data was collected? (15 months)

REQUIREMENTS DOCUMENT EVALUATION
Product Questions

Changes/defects:

How many changes are there to the document? (88)

How many of the changes are errors? (79)

What is the distribution of errors in the requirements document by type of misunderstanding (I.e., **ambiguity, omission, inconsistency, incorrect fact, wrong section**)?

Context:

How is the document being used?

How was the need for change discovered?

REQUIREMENTS DOCUMENT EVALUATION
Product Questions

Quality perspective: Ease of change

Cost:

What is the distribution of the types of changes and effort to make them?
What is the distribution of changes by staff time to make the changes?
Is the effort to change the document low?

Document well-structuredness:

Are most of the changes were confined to one section of the document?

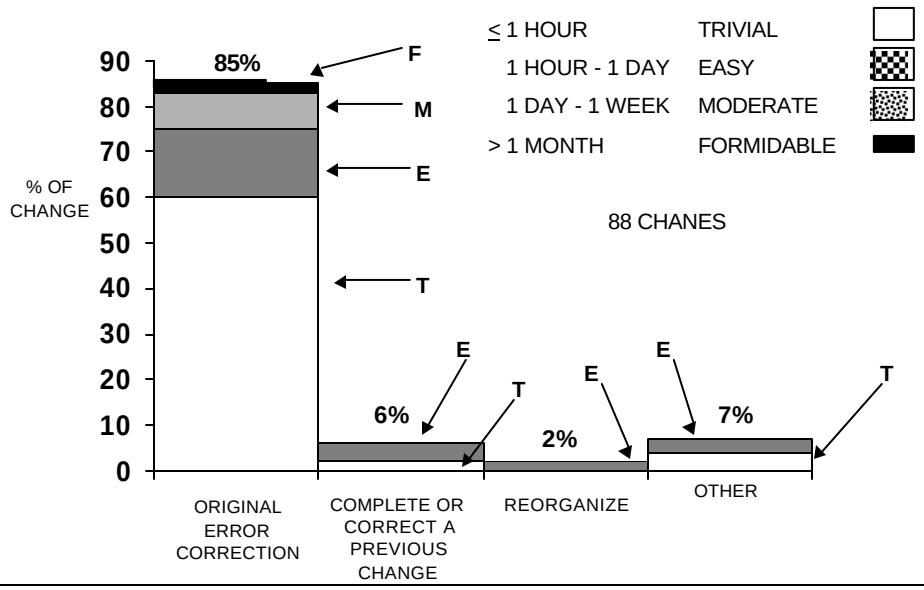
Consistency and Unambiguity:

How consistent and unambiguous is the document relative to its precision and completeness?

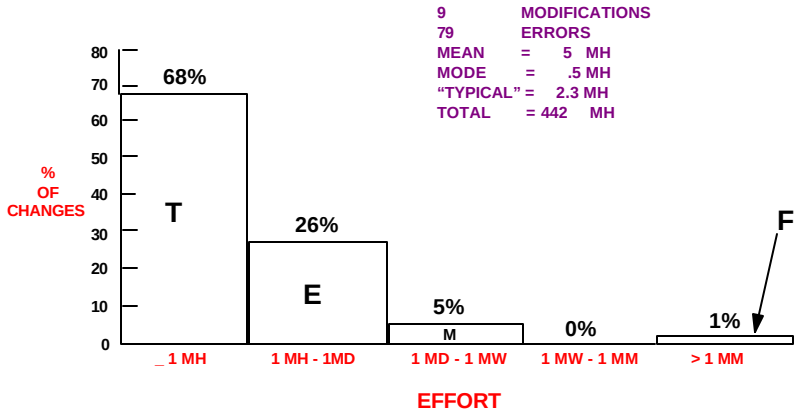
Worthiness of being maintained:

How is the document being used?Is it used in important and relevant ways?

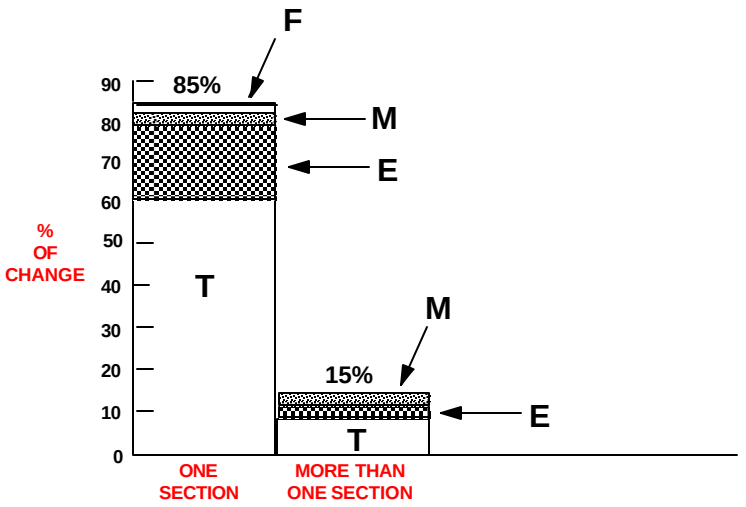
TYPES OF CHANGES



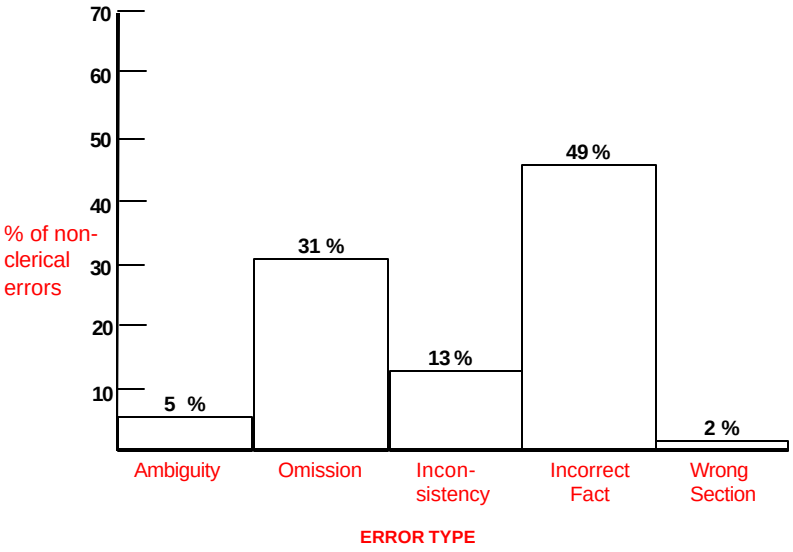
EFFORT TO CHANGE



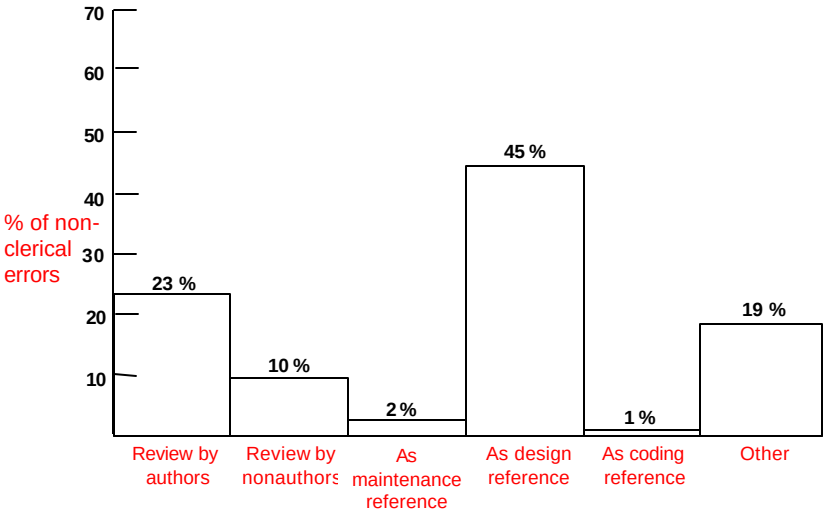
CONFINEMENT OF CHANGES



NONCLERICAL ERRORS BY TYPE



USE OF DOCUMENT
Discovery of Need for Change



REQUIREMENTS DOCUMENT EVALUATION
~~Conclusions~~

DATA COLLECTION METHODOLOGY FEEDBACK

Partial data provides useful feedback to developers

Data analysis generates new questions of interest

A-7 REQUIREMENTS DOCUMENT FEEDBACK

The document is relatively more consistent and precise than complete and correct.

Most changes are confined to single sections
Can we conclude the document is well-structured?

Seems to be a small effort to make changes
Can we conclude the document is easy to change (maintain)?

The document is heavily used as a design reference
Can we conclude that it is worth maintaining?

REQUIREMENTS DOCUMENT EVALUATION
Conclusions

Analyze the **defects** in order to **characterize** them with respect to the **various classification schemes** from the point of view of the **knowledge builder** .in the context of

a requirements document development, ...

Can classify requirements faults according to omission, incorrect fact, ambiguity, inconsistency, …

BUILDING DEFECT BASELINES

Error Origin Classification

Goal:

Analyze the **life cycle process for a class of projects** to **characterize it** with respect to **error origin** from the point of view of **quality assurance baselining**.

Environment:

- NASA/GSFC, SEL
- Ground support software for unmanned spacecraft control
- 50K to 120K source lines of Fortran code
- Design through acceptance test

Experimental design:

- Multi-project variation
- Programmers/managers from the same population

BUILDING DEFECT BASELINES

Error Origin Classification

Questions:

Process conformance:

What is the life cycle model?

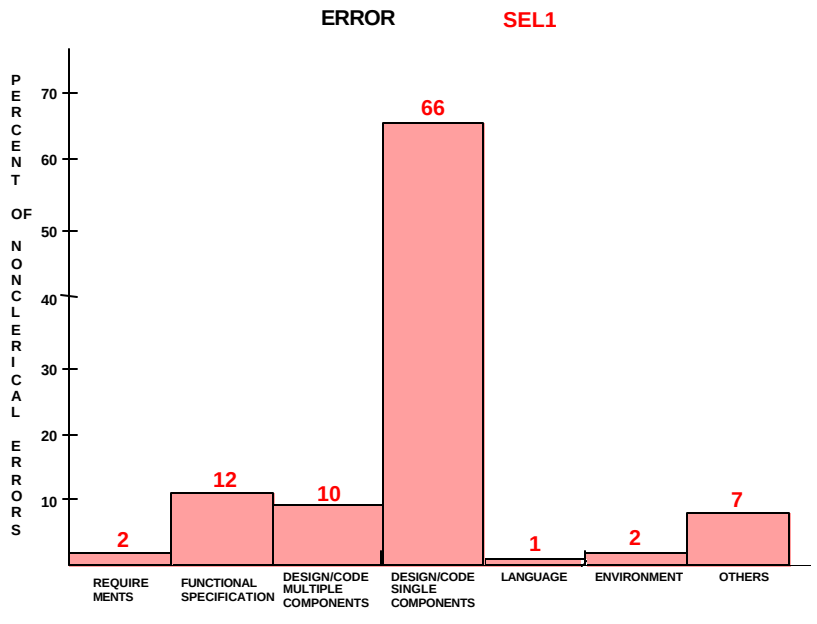
How well is it being applied?

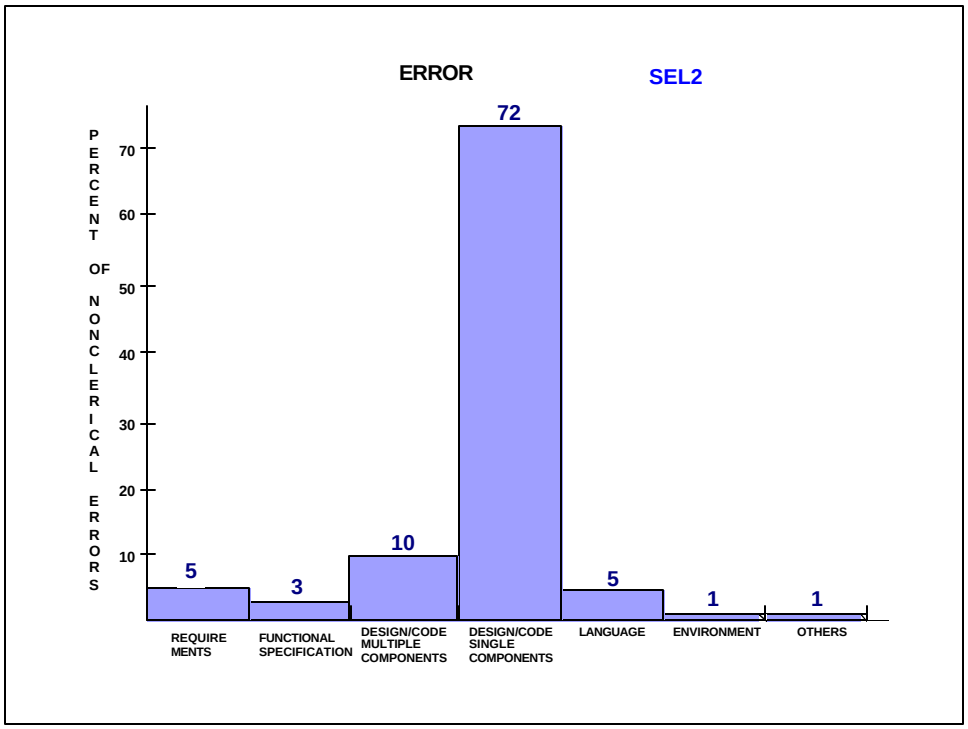
Domain conformance:

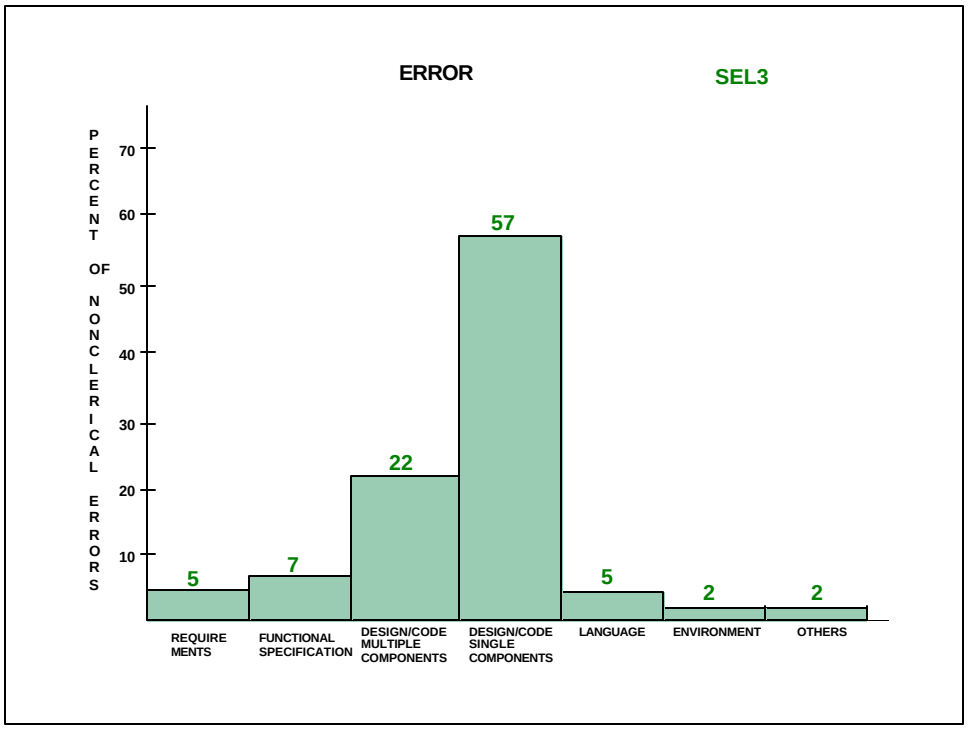
How well do the developers understand the application?

Quality perspective:

What is the distribution of errors by error origin (i.e., according to the misunderstanding that caused them)?




ERROR
SEL2
PERCENT OF NONCLERICAL ERRORS
70
60
50
40
30
20
10
72
5
3
10
5
1
1
REQUIRE MENTS
FUNCTIONAL SPECIFICATION
DESIGN/CODE MULTIPLE COMPONENTS
DESIGN/CODE SINGLE COMPONENTS
LANGUAGE
ENVIRONMENT
OTHERS


ERROR
SEL3
PERCENT OF NONCLERICAL ERRORS
70
60
50
40
30
20
10
57
22
5
7
5
2
2
REQUIRE MENTS
FUNCTIONAL SPECIFICATION
DESIGN/CODE MULTIPLE COMPONENTS
DESIGN/CODE SINGLE COMPONENTS
LANGUAGE
ENVIRONMENT
OTHERS

BUILDING DEFECT BASELINES

SIMULATOR FOR SATELLITE PLANNING STUDIES

Goals

Analyze the life cycle defects for a particular project in order to

characterize them with respect to various error, fault, and failure classes

evaluate them with respect to those from other studies

characterize them with respect to the relationship between errors and complexity

from the point of view of the experience factory

Basili/Perricone

PROJECT BACKGROUND

- **General purpose program for satellite planning studies**

Size: 90K Source lines and 517 Code segments

370 FORTRAN Subroutines, 36 Assembly segments, 111 COMMON modules, BLOCK data, Utity routines

Modified modules - Adopted from a previous system (72%)

New modules - Developed specifically for this system

- **Requirements for the system kept growing and changing over the life cycle**

Defects Two definitions - Faults[Textual](215) and Errors [Conceptual] (155)

49% Defects/faults in modified modules

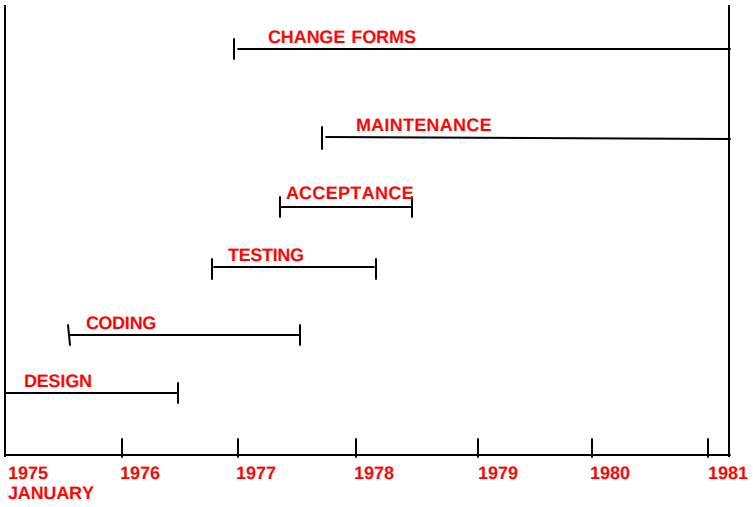
51% Defects/faults in new modules

Corrections vs Modifications

38% of changes were modifications

62% of changes were fault corrections

LIFE CYCLE OF ANALYZED SOFTWARE



NUMBER MODULES

	ALL MODULES		MODULES WITH FAULTS	
NUMBER OF LINES	SOURCE	EXECUTABLE	SOURCE	EXECUTABLE
0-50	50	258	3	49
51-	107	70	16	25
101-150	80	26	20	13
151-200	56	13	19	7
201-250	34	1	12	1
251-300	14	1	9	0
301-350	7	1	4	1
351-400	9	0	7	0
>400	10	0	6	0
TOTAL	370	370	96	96

OF MODULES AFFECTED BY AN ERROR
Faults:

"211 Texual Errors 174 Conceptual Errors)

# ERRORS		# MODULES AFFECTED
155	(89%)	1
9		2
3		3
6		4
1		5

RESULTS: SIMILAR TO OTHER STUDIES, FEW ERRORS INVOLVE MORE THAN ONE MODULE

NUMBER OF ERRORS PER MODULE
(Faults: 215 Faults)

# Modules	New	Modified	Errors / Module
36	17	19	1
26	13	13	2
16	10	6	3
13	7	6	4
4	1**	3*	5
1	1**		7

NEW MODULES
Effort to Correct Faults in the Most Fault-Prone New Model

	NUMBER OF ERRORS (12 TOTAL)	AVERAGE EFFORT TO CORRECT
MISUNDERSTOOD OR INCORRECT REQUIREMENTS	8	32 HOURS
INCORRECT DESIGN OR IMPLEMENTATION OF A MODULE	3	0.5 HOURS
CLERICAL ERROR	1	0.5 HOURS

NEW MODULES
Effort to Correct Faults in the Most Fault-Prone Modified Model

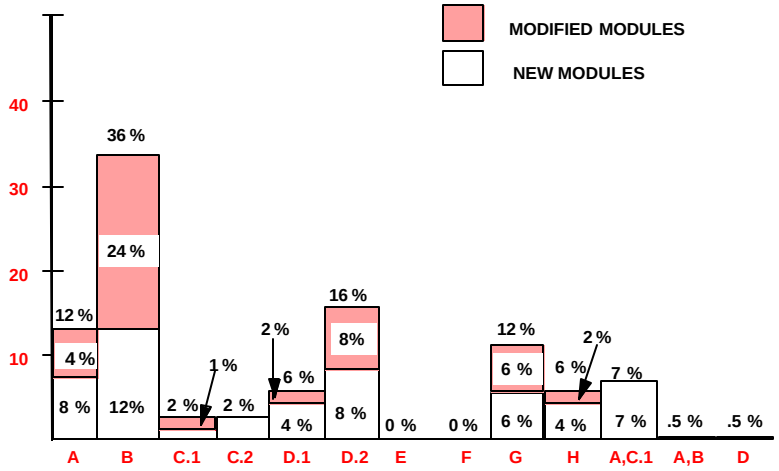
	NUMBER OF ERRORS (15 TOTAL)	AVERAGE EFFORT TO CORRECT
MISUNDERSTOOD OR INCORRECT REQUIREMENTS	8	24 HOURS
INCORRECT DESIGN OR IMPLEMENTATION OF A MODULE COMPONENT	5	16 HOURS
CLERICAL ERROR	2	4.5 HOURS

ERROR DISTRIBUTION BY TYPE

CATEGORIES:

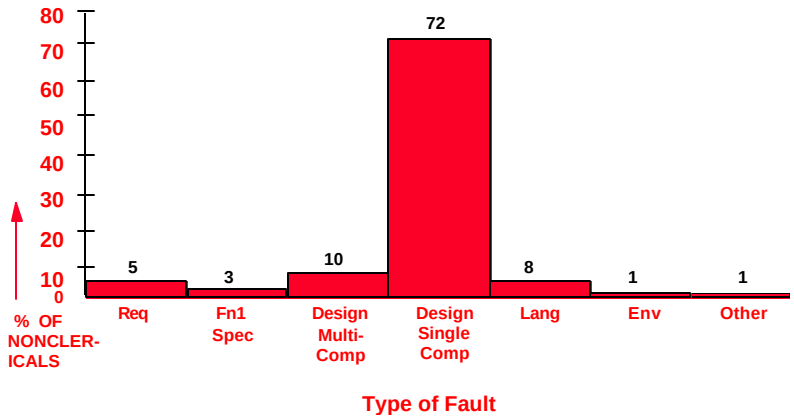
- **A:** REQUIREMENTS INCORRECT OR MISINTERPRETED
- **B:** FUNCTIONAL SPECIFICATION INCORRECT OR MISINTERPRETED
- **C:** DESIGN ERROR INVOLVING SEVERAL COMPONENTS
- **D:** DESIGN ERROR IN A SINGLE COMPONENTS
- **E:** MISUNDERSTANDING OF EXTERNAL ENVIRONMENT
- **F:** ERRORS IN PROGRAMMING LANGUAGE OR COMPILER
- **G:** CLERICAL ERROR
- **H:** ERROR DUE TO PREVIOUS MISCORRECTION OF AN ERROR

SOURCE OF ERRORS

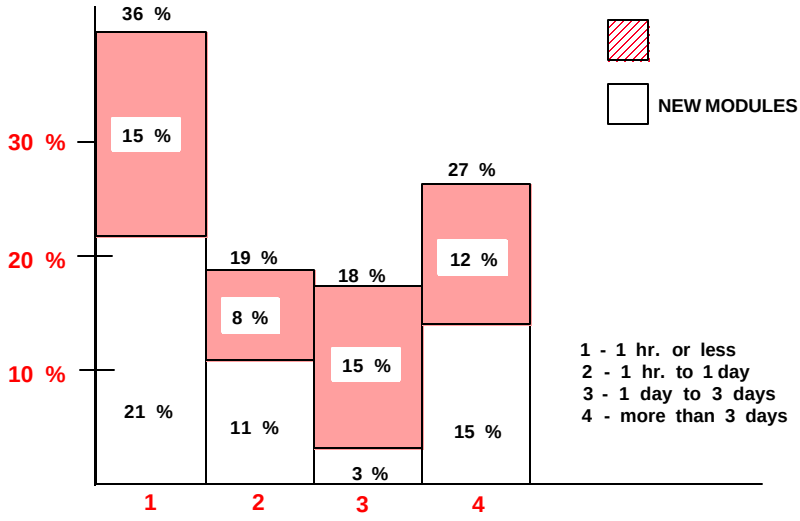


RESULTS: SIMILAR TO ENDRES' STUDY (46 % vs 48 % HERE INVOLVED MISUNDERSTANDING OF THE PROBLEM

SEL2 SOURCES OF NONCLERICAL ERRORS



EFFORT



FAULT TYPES

CATEGORIES

INITIALIZATION - failure to initialize data on entry or exit

CONTROL STRUCTURE - incorrect path taken

INTERFACE - associated with structure outside module's environment

DATA - incorrect use of a data structure

COMMISSION - incorrect executable statement

OMISSION - neglecting to include some entity in a module

(Con't)

CLASSIFICATION OF FAULTS

	COMMISSION		OMISSION	
	NEW	MODIFIED	NEW	MODIFIED
INITIALIZATION	2	9	5	9
CONTROL	12	2	16	6
INTERFACE	23	31	27	6
DATA	10	17	1	3
COMPUTATION	16	21	3	3
	28 %	36 %	23 %	12 %
	64 %		35 %	

	TOTAL	
	NEW	MODIFIED
INITIALIZATION	7	18 - - - 25 (11%)
CONTROL	28	8 - - - 36 (16%)
INTERFACE	50	37 - - - 87 (39%)
DATA	11	20 - - - 31 (14%)
COMPUTATION	19	24 - - - 43 (19%)
	115	107

FAULT TYPES

RESULT:

The largest percent of faults involve interface (39%)
Control is more of a problem in new modules
Data and initialization are more of a problem in modified modules
Small number of omission faults in modified modules

POSSIBLE EXPLANATION

– The basic algorithms for the modified modules were correct but needed some adjustment with respect to data values and initialization for the application of the old algorithm to the new application

FAULTS/1000 EXECUTABLE LINES (INCLUDES ALL MODULES)

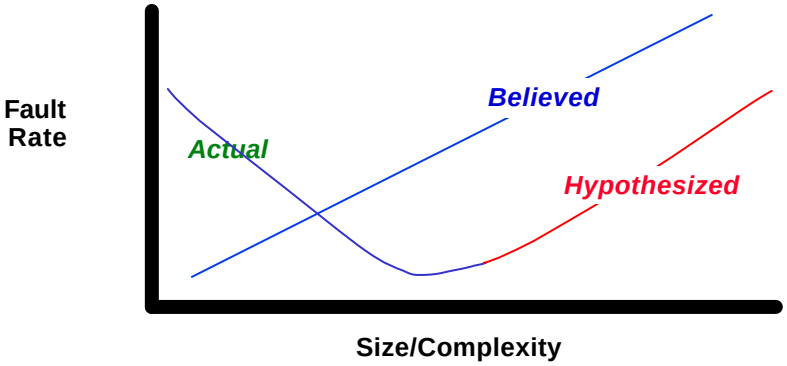
MODULE SIZE	FAULTS/1000 LINES
50	16.0
100	12.6
150	12.4
200	7.6
>200	6.4

POSSIBLE EXPLANATIONS:

Interface faults are spread across all modules
The majority of modules examined were small, biasing the result
The larger modules were coded with more care
The faults in smaller modules were more apparent

Fault Rate vs. Size

Measuring Fault Rate against Size and Complexity



AVERAGE CYCLOMATIC COMPLEXITY FOR ALL MODULES

MODULE SIZE	AVERAGE CYCLOMATIC COMPLEXITY
50	6.0
100	17.9
150	28.1
200	52.7
>200	60.0

COMPLEXITY AND ERROR RATE FOR ERRORED MODULES

Module Size	Average Cyclomatic Complexity	Faults/1000 Executable Lines
50	6.2	65.0
100	19.6	33.3
150	27.5	24.6
200	56.7	13.4
>200	77.5	9.7

- **RESULT:**
- Average cyclomatic complexity grew faster than size

SUMMARY

Defect Analysis provides useful information
- Can see new application with changing requirements
- Fault profile for new and modified modules different
- Fault profile for a new application different than that of more mature application

Module size an open issues with respect to fault rate
- evidence that lager modules, within limits, may be less fault prone
- should not put artificial limits on module size

BUILDING DEFECT BASELINES
Conclusions

The majority of faults are made during the design and implementation of a single component

There is a pattern to the origin of defects for a particular project class in a particular environment

Analyze the **defects** in order to **characterize** them with respect to the **various classification schemes** from the point of view of the **knowledge builder** .in the context of

a requirements document development, ...

BUILDING DEFECT BASELINES

Inspection Process Fault Classification

Goal:

Analyze the **inspection process** in order to **characterize** it with respect to the **fault correction effort** from the point of view of the **manager**

Environment:

- Major mainframe manufacturer
- Next release of a library tool
- Development or modification of 40K source lines
- Total size 100K SLOC
- PL/1 like language

Questions:

- What was the isolation and fix effort and total error correction effort for errors of omission and commission?

Selby/Basili

BUILDING DEFECT BASELINES
Inspection Process Fault Characterization

Fault Correction Effort in hours by fault class

Average Effort	Commission	Omission	All
Isolation Effort	7.2	4.2	6.3
Fix Effort	3.7	3.9	3.7
Total Correction Effort	10.9	8.1	10.0

Isolating a fault took almost twice as much effort as fixing it
Correcting a fault of commission required more effort than a fault of omission
Isolating a fault of commission required twice the effort as a fault of omission

Note: Could count effort for fixing a missing error as development effort

BUILDING DEFECT BASELINES
Inspection Process Fault Characterization

Conclusions

During design, it is less costly to leave out a design/code segment than to include an incorrect one

Analyze the **defects** in order to **characterize** them with respect to the **various classification schemes** from the point of view of the **knowledge builder** .in the context of

faults collected during an inspection process, ...

Investigating Influential Factors for Software Process Improvement

Goal

Analyze the **project set** in order to **evaluate and improve** future systems with respect to the **defect prevention and detection** from the point of view of the **organization**

Environment:

Matsushita Communications, four communications software projects
Waterfall model development, collected during acceptance test and operation

Experimental design:

multi-project study
in vivo, experienced subjects

Mashiko/Basili

Investigating Influential Factors for Software Process Improvement

Definitions

Defect notation: $D_n(d1_n,d2_n,d3_n,d4_n;Cd_n,SD_n,Md_n)$, where
d1 injection phase: when defect entered system
d2detection phase: when defect was found
d3 error type: Logic, Communication, Omission, Commission
d4 fault type: see table 1
Cd cost of defect: cost to fix
Sd severity of defect: see table 2
Md number of modules affected

Investigating Influential Factors for Software Process Improvement

Table 1. Fault Type

Fault type	Abbreviation	Definition
Global structure	Gstr'	Fault of relationships among subsystems
Data structure	Dstr	Fault of structure of files, tables or other data, including fault of data size
Algorithm	Algr	Fault of algorithm inside program module
Human interface	Hitr	Fault of human interface
External interface	Eitr	Fault of interface between the product and its external system
Internal interface	Iitr	Fault of interface between modules
Initialization	Init	Omission or Commission of initialization of data entry
Constant value	Cnst	Fault of definition of constant value

Investigating Influential Factors for Software Process Improvement

Table 2. Severity of Defect

Level	Name	Definition
4	Critical	Without fixing a defect of this level, delivery of the product to client is impossible.
3	Essential	Without fixing a defect of this level, operation is possible by altering normal operational procedure for the system. It must be fixed as soon as possible.
2	Important	Without fixing a defect of this level, normal operation is possible. However, the efficiency of operation is improved by fixing it.
1	Desired	A defect of this level does not cause trouble to the efficiency of operation. However, it is desirable to fix it from the point of view of the product's impression to the user.

Some Results

There was a pattern of effects for one class of projects (3) with similar characteristics

Errors of omission represent at least 40% of the defects

Highest proportion of severe defects caused by external interface faults

A fault caused by omission is more costly to fix than a defect caused by commission

A fault injected during the requirements phase costs at least 44% more to fix than a fault injected during any other phase

Knowledge Packaging

What have we learned from the set of defect studies so far?

There is value in multiple studies for both supporting and not supporting hypotheses

Make sure you are comparing like things, e.g., same classification (injection time, detection time, environment, subjects, phase of data collection)

Vary the classification to check the effects along the various values

There are insights to be gained from the collection and analysis of defects according to different classification schemes, independent of the scheme

Hypotheses:

Are there classes of omission that affect the cost of fix? Or are there certain applications that cost more to modify? What is the relationship of the architecture to the cost of the modification?

Knowledge Packaging

What have we learned from the set of defect studies so far?

Hypotheses:

Defect rate goes down as size, complexity rise (within limits)

Modified and new modules have different defect profiles

Omission defects are difficult to fix after delivery but easy to fix during design

Errors injected in the requirements phase are more expensive to fix than those injected in other phases, when detected at a later phase

Consequences:

The techniques used to build reusable models should be tailored based upon the anticipated defect classes

Iterative development is better suited o minimizing the cost of defect fixing (lower cost of omission faults found early)

FUNCTIONAL TEST PLAN EVALUATION

Goal

Analyze the **acceptance test plan** in order to **evaluate and improve** it for future releases with respect to the **ability of the acceptance test suite to cover the operational use of the system** from the point of view of the **test developer**

Environment:

NASA SEL

Subset of a large satellite system

Experimental design:

Single project/case study

in vitro, no subjects

FUNCTIONAL TEST PLAN EVALUATION

Process Questions

Process conformance:

What is the test methodology?
(standard methodology used in the SEL)

How well is it being applied?
(testers have used the methodology before)

Domain conformance:

How well do the testers understand the application?
(reasonably well)

FUNCTIONAL TEST PLAN EVALUATION

Product Questions

Product dimensions:

What is the size of the system?
(68 Fortran subroutines, 10,000 lines of code
4,300 executable statements)

What is the size of the test suite?
(10 multi-part acceptance tests,
not a rigorous sampling of input domain but not trivial)

What are the number of operational uses? (60 uses)

Changes/defects:

How many faults were found during acceptance test?

How many faults were found during operational use? (8)

Context:

How was the system being used during operation?
(normal use)

FUNCTIONAL TEST PLAN EVALUATION

Product Questions

Quality perspective: Compare the structural coverage of the acceptance tests and operational use of the system.

What is the procedure coverage for the acceptance test suite by test and in total?

What is the statement coverage for the acceptance test suite by test and in total?

What is the % of unique code exercised by each test?

What is the procedure coverage for the operational use of the system?

What is the statement coverage for the operational use of the system?

What is the overlap of the acceptance test and operational use coverage?

Is there anything different about the statements executed in operational test but not covered during acceptance test?

FUNCTIONAL TEST PLAN EVALUATION

Product Questions

Feedback:

Is there any indication, based upon the coverage representation, to indicate whether reliability models can be applied during acceptance test to predict operational reliability?

FUNCTIONAL TEST PLAN EVALUATION
Structural Coverage of Acceptance Test

Executable Statement Coverage
by 10 TestCases

Case	Procedures Executed (%)	Executable Statements (%)	% Unique Code
t1	50.0	27.5	0.0
t2	50.0	27.2	0.0
t3	48.5	24.4	0.0
t4	60.3	37.9	4.4
t5	69.1	47.1	1.7
t6	67.6	42.7	0.0
t7	66.2	39.0	0.0
t8	66.2	45.6	1.0
t9	66.2	41.0	0.0
t10	66.2	40.2	0.0
Cumulative	75.0	56.0	
Intersect	42.6	18.1	

44% of executable statements were not exercised in acceptance test.
They may have been executed in system/unit testing

FUNCTIONAL TEST PLAN EVALUATION
~~Structural Coverage of Operational Use~~

Structural Coverage of
60 Operational Usage Cases

	Procedures Executed (%)	Executed Statements (%)
Cumulative	80.0	64.9
Intersection	27.9	10.3

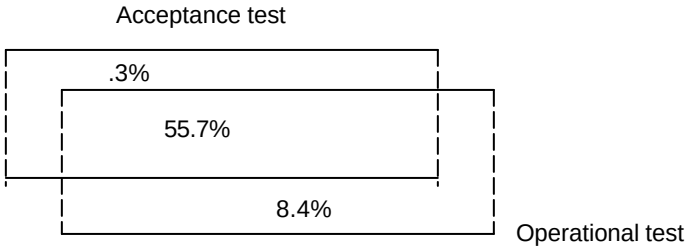
10% of the code was executed by all of the operational cases

FUNCTIONAL TEST PLAN EVALUATION

Are Acceptance Tests representative of operational usage?

Must be true if acceptance test failures used to predict operational failures

Coverage:



Representation:

The mix of statements in the 8.4% and 55.7% differ

Twice as likely to execute a call or if in the 8.4%

Otherwise can't distinguish by structural coverage numbers

However, no faults were revealed in the 8.4%

FUNCTIONAL TEST PLAN EVALUATION
Observations

Functional test plan reasonably effective, but could be refined for future releases.

About 56% of code exercised by acceptance tests; 65% by operational use.

Acceptance test reasonably representative of operational tests, no faults found in unexercised code.

If acceptance tests randomized, reliability models may be used to predict operational reliability with moderate chance of success.