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
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

<table>
<thead>
<tr>
<th>Number of Defects</th>
<th>Number of Modules Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>371 (85%)</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
## Defect Distribution by Modules

<table>
<thead>
<tr>
<th>Number of Defects</th>
<th>Number of Modules Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>371 (85%)</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
# DEFECT CHARACTERIZATION

## Defect Distribution by Modules

**Number of Defects per Module**

<table>
<thead>
<tr>
<th>Number of Modules</th>
<th>Number of Defects/Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>28</td>
</tr>
</tbody>
</table>
# DEFECT CHARACTERIZATION

## Problem Specific Defects

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Configuration and Architecture</td>
<td>10</td>
</tr>
<tr>
<td>Dynamic Behavior and Communication between Processes</td>
<td>17</td>
</tr>
<tr>
<td>Functions Offered</td>
<td>12</td>
</tr>
<tr>
<td>Output Listings and Formats</td>
<td>3</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>3</td>
</tr>
<tr>
<td>Performance</td>
<td>1</td>
</tr>
</tbody>
</table>

46%
# DEFECT CHARACTERIZATION

## Implementation Specific Defects

<table>
<thead>
<tr>
<th>Defect</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization (of Fields and Areas)</td>
<td>8</td>
</tr>
<tr>
<td>Addressability (in the sense of the assembler)</td>
<td>7</td>
</tr>
<tr>
<td>Reference to Names</td>
<td>7</td>
</tr>
<tr>
<td>Counting and Calculating</td>
<td>8</td>
</tr>
<tr>
<td>Masks and Comparisons</td>
<td>2</td>
</tr>
<tr>
<td>Estimation of Range Limits (addresses and parameters)</td>
<td>1</td>
</tr>
<tr>
<td>Placing of instructions within a module (bad fixes)</td>
<td>5</td>
</tr>
</tbody>
</table>

*Total: 38%*
<table>
<thead>
<tr>
<th>Textual Defects</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spelling in messages and commentaries</td>
<td>4</td>
</tr>
<tr>
<td>Missing commentaries or flowcharts</td>
<td>5</td>
</tr>
<tr>
<td>(standards)</td>
<td></td>
</tr>
<tr>
<td>Incompatible status of macros or modules</td>
<td>5</td>
</tr>
<tr>
<td>(integration defects)</td>
<td></td>
</tr>
<tr>
<td>Not classifiable</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>16%</td>
</tr>
</tbody>
</table>
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**.
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)
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)
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 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?
EFFORT TO CHANGE

9 MODIFICATIONS
79 ERRORS
MEAN = 5 MH
MODE = .5 MH
“TYPICAL” = 2.3 MH
TOTAL = 442 MH

% OF CHANGES

68% 26% 5% 0% 1%

EFFORT
CONFINEMENT OF CHANGES

% OF CHANGE

ONE SECTION

MORE THAN ONE SECTION

85%

15%
NONCLERICAL ERRORS BY TYPE

% of non-clerical errors

- Ambiguity: 5%
- Omission: 31%
- Inconsistency: 13%
- Incorrect Fact: 49%
- Wrong Section: 2%

ERROR TYPE
USE OF DOCUMENT
Discovery of Need for Change

% of non-clerical errors

- Review by authors: 23%
- Review by nonauthors: 10%
- As maintenance reference: 2%
- As design reference: 45%
- As coding reference: 1%
- Other: 19%
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?
G1: 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, ...

G2: Analyze the requirements/specification process (SCR) in order to evaluate it with respect to the ease of modification and quality of the requirements/specification document produced from the point of view of the knowledge builder. In the context of a requirements document development, ...

Issues: hard to evaluate without any baselines
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)?
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

SIMULATOR FOR SATELLITE PLANNING STUDIES

Goals

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

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

evaluate them with respect to those from other studies

categorize 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, Utility 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


- DESIGN
- CODING
- TESTING
- ACCEPTANCE
- MAINTENANCE
- CHANGE FORMS
## NUMBER MODULES

<table>
<thead>
<tr>
<th>NUMBER OF LINES</th>
<th>ALL MODULES</th>
<th></th>
<th>MODULES WITH FAULTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SOURCE</td>
<td>EXECUTABLE</td>
<td>SOURCE</td>
<td>EXECUTABLE</td>
</tr>
<tr>
<td>0-50</td>
<td>50</td>
<td>258</td>
<td>3</td>
<td>49</td>
</tr>
<tr>
<td>51-</td>
<td>107</td>
<td>70</td>
<td>16</td>
<td>25</td>
</tr>
<tr>
<td>101-150</td>
<td>80</td>
<td>26</td>
<td>20</td>
<td>13</td>
</tr>
<tr>
<td>151-200</td>
<td>56</td>
<td>13</td>
<td>19</td>
<td>7</td>
</tr>
<tr>
<td>201-250</td>
<td>34</td>
<td>1</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>251-300</td>
<td>14</td>
<td>1</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>301-350</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>351-400</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>&gt;400</td>
<td>10</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

**TOTAL**

370  370  96  96
# OF MODULES AFFECTED BY AN ERROR

Faults:

“211 Textual Errors 174 Conceptual Errors"

<table>
<thead>
<tr>
<th># ERRORS</th>
<th># MODULES AFFECTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>155 (89%)</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

RESULTS: SIMILAR TO OTHER STUDIES, FEW ERRORS INVOLVE MORE THAN ONE MODULE
<table>
<thead>
<tr>
<th># Modules</th>
<th>New</th>
<th>Modified</th>
<th>Errors / Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>17</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>26</td>
<td>13</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1**</td>
<td>3*</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1**</td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>
## Effort to Correct Faults in the Most Fault-Prone New Model

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Number of Errors</th>
<th>Average Effort to Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misunderstood or Incorrect Requirements</td>
<td>8</td>
<td>32 hours</td>
</tr>
<tr>
<td>Incorrect Design or Implementation of a Module</td>
<td>3</td>
<td>0.5 hours</td>
</tr>
<tr>
<td>Clerical Error</td>
<td>1</td>
<td>0.5 hours</td>
</tr>
</tbody>
</table>
# Effort to Correct Faults in the Most Fault-Prone Modified Model

<table>
<thead>
<tr>
<th>Error Description</th>
<th>Number of Errors</th>
<th>Average Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misunderstood or incorrect requirements</td>
<td>8</td>
<td>24 hours</td>
</tr>
<tr>
<td>Incorrect design or implementation of a module</td>
<td>5</td>
<td>16 hours</td>
</tr>
<tr>
<td>Clerical error</td>
<td>2</td>
<td>4.5 hours</td>
</tr>
</tbody>
</table>

(15 total errors)
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
SOURCES OF ERRORS

<table>
<thead>
<tr>
<th>Type of Fault</th>
<th>% of Nonclericals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Req</td>
<td>15</td>
</tr>
<tr>
<td>Fn1 Spec</td>
<td>40</td>
</tr>
<tr>
<td>Design Multi-Comp</td>
<td>4</td>
</tr>
<tr>
<td>Design Single Comp</td>
<td>22</td>
</tr>
<tr>
<td>Lang</td>
<td>0</td>
</tr>
<tr>
<td>Env</td>
<td>0</td>
</tr>
<tr>
<td>Other</td>
<td>18</td>
</tr>
</tbody>
</table>
SEL2 SOURCES OF NONCLERICAL ERRORS

Type of Fault

- Req: 5
- Fn1 Spec: 3
- Design Multi-Comp: 10
- Design Single Comp: 72
- Lang: 8
- Env: 1
- Other: 1

% of Nonclerical Errors
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

<table>
<thead>
<tr>
<th></th>
<th>COMMISSION</th>
<th></th>
<th>OMISSION</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NEW</td>
<td>MODIFIED</td>
<td>NEW</td>
<td>MODIFIED</td>
</tr>
<tr>
<td>INITIALIZATION</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>CONTROL</td>
<td>12</td>
<td>2</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>23</td>
<td>31</td>
<td>27</td>
<td>6</td>
</tr>
<tr>
<td>DATA</td>
<td>10</td>
<td>17</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>COMPUTATION</td>
<td>16</td>
<td>21</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td><strong>28 %</strong></td>
<td><strong>36 %</strong></td>
<td><strong>23 %</strong></td>
<td><strong>12 %</strong></td>
</tr>
<tr>
<td></td>
<td><strong>64 %</strong></td>
<td></td>
<td></td>
<td><strong>35 %</strong></td>
</tr>
</tbody>
</table>

### TOTAL

<table>
<thead>
<tr>
<th></th>
<th>NEW</th>
<th>MODIFIED</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALIZATION</td>
<td>7</td>
<td>18 - - - 25 (11%)</td>
</tr>
<tr>
<td>CONTROL</td>
<td>28</td>
<td>8 - - - 36 (16%)</td>
</tr>
<tr>
<td>INTERFACE</td>
<td>50</td>
<td>37 - - - 87 (39%)</td>
</tr>
<tr>
<td>DATA</td>
<td>11</td>
<td>20 - - - 31 (14%)</td>
</tr>
<tr>
<td>COMPUTATION</td>
<td>19</td>
<td>24 - - - 43 (19%)</td>
</tr>
<tr>
<td></td>
<td><strong>115</strong></td>
<td></td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>MODULE SIZE</th>
<th>FAULTS/1000 LINES</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>16.0</td>
</tr>
<tr>
<td>100</td>
<td>12.6</td>
</tr>
<tr>
<td>150</td>
<td>12.4</td>
</tr>
<tr>
<td>200</td>
<td>7.6</td>
</tr>
<tr>
<td>&gt;200</td>
<td>6.4</td>
</tr>
</tbody>
</table>

### 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
Measuring Fault Rate against Size and Complexity

Fault Rate vs. Size

- Actual
- Believed
- Hypothesized

Fault Rate

Size/Complexity
### Average Cyclomatic Complexity for All Modules

<table>
<thead>
<tr>
<th>Module Size</th>
<th>Average Cyclomatic Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>6.0</td>
</tr>
<tr>
<td>100</td>
<td>17.9</td>
</tr>
<tr>
<td>150</td>
<td>28.1</td>
</tr>
<tr>
<td>200</td>
<td>52.7</td>
</tr>
<tr>
<td>&gt;200</td>
<td>60.0</td>
</tr>
</tbody>
</table>
# COMPLEXITY AND ERROR RATE FOR ERRORED MODULES

<table>
<thead>
<tr>
<th>Module Size</th>
<th>Average Cyclomatic Complexity</th>
<th>Faults/1000 Executable Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>6.2</td>
<td>65.0</td>
</tr>
<tr>
<td>100</td>
<td>19.6</td>
<td>33.3</td>
</tr>
<tr>
<td>150</td>
<td>27.5</td>
<td>24.6</td>
</tr>
<tr>
<td>200</td>
<td>56.7</td>
<td>13.4</td>
</tr>
<tr>
<td>&gt;200</td>
<td>77.5</td>
<td>9.7</td>
</tr>
</tbody>
</table>

- **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 larger modules, within limits, may be less fault prone
  ➔ should not put artificial limits on module size
BUILDING DEFECT BASELINES
Study Conclusions

Different project characteristics/environments produce different patterns of defect origin.

Projects with new requirements have more defects traceable to the requirement/specification phase.

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 full life cycle at NASA/SEL.

Classification schemes:
- omission, commission
- interface, control flow, data (initialization, use)
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

<table>
<thead>
<tr>
<th>Average Effort</th>
<th>Commission</th>
<th>Omission</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolation Effort</td>
<td>7.2</td>
<td>4.2</td>
<td>6.3</td>
</tr>
<tr>
<td>Fix Effort</td>
<td>3.7</td>
<td>3.9</td>
<td>3.7</td>
</tr>
<tr>
<td>Total Correction</td>
<td>10.9</td>
<td>8.1</td>
<td>10.0</td>
</tr>
</tbody>
</table>

- 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
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,
- Defect data collected during acceptance test and operation

**Experimental design:**
- multi-project study
- in vivo, experienced subjects
Investigating Influential Factors for Software Process Improvement

Definitions

Defect notation: $D_n(d_{1n}, d_{2n}, d_{3n}, d_{4n}; C_{dn}, S_{dn}, M_{dn})$, where

- $d_1$ injection phase: when defect entered system
- $d_2$ detection phase: when defect was found
- $d_3$ error type: Logic, Communication, Omission, Commission
- $d_4$ fault type: see table 1
- $C_d$ cost of defect: cost to fix
- $S_d$ severity of defect: see table 2
- $M_d$ number of modules affected
### Investigating Influential Factors for Software Process Improvement

#### Table 1. Fault Type

<table>
<thead>
<tr>
<th>Fault type</th>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global structure</td>
<td>Gstr'</td>
<td>Fault of relationships among subsystems</td>
</tr>
<tr>
<td>Data structure</td>
<td>Dstr</td>
<td>Fault of structure of files, tables or other data, including fault of data size</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Algr</td>
<td>Fault of algorithm inside program module</td>
</tr>
<tr>
<td>Human interface</td>
<td>Hitr</td>
<td>Fault of human interface</td>
</tr>
<tr>
<td>External interface</td>
<td>Eitr</td>
<td>Fault of interface between the product and its external system</td>
</tr>
<tr>
<td>Internal interface</td>
<td>Iitr</td>
<td>Fault of interface between modules</td>
</tr>
<tr>
<td>Initialization</td>
<td>Init</td>
<td>Omission or Commission of initialization of data entry</td>
</tr>
<tr>
<td>Constant value</td>
<td>Cnst</td>
<td>Fault of definition of constant value</td>
</tr>
</tbody>
</table>
# Investigating Influential Factors for Software Process Improvement

## Table 2. Severity of Defect

<table>
<thead>
<tr>
<th>Level</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Critical</td>
<td>Without fixing a defect of this level, delivery of the product to client is impossible.</td>
</tr>
<tr>
<td>3</td>
<td>Essential</td>
<td>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.</td>
</tr>
<tr>
<td>2</td>
<td>Important</td>
<td>Without fixing a defect of this level, normal operation is possible. However, the efficiency of operation is improved by fixing it.</td>
</tr>
<tr>
<td>1</td>
<td>Desired</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 to 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
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**

<table>
<thead>
<tr>
<th>Case</th>
<th>Procedures Executed (%)</th>
<th>Executable Statements (%)</th>
<th>% Unique Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>50.0</td>
<td>27.5</td>
<td>0.0</td>
</tr>
<tr>
<td>t2</td>
<td>50.0</td>
<td>27.2</td>
<td>0.0</td>
</tr>
<tr>
<td>t3</td>
<td>48.5</td>
<td>24.4</td>
<td>0.0</td>
</tr>
<tr>
<td>t4</td>
<td>60.3</td>
<td>37.9</td>
<td>4.4</td>
</tr>
<tr>
<td>t5</td>
<td>69.1</td>
<td>47.1</td>
<td>1.7</td>
</tr>
<tr>
<td>t6</td>
<td>67.6</td>
<td>42.7</td>
<td>0.0</td>
</tr>
<tr>
<td>t7</td>
<td>66.2</td>
<td>39.0</td>
<td>0.0</td>
</tr>
<tr>
<td>t8</td>
<td>66.2</td>
<td>45.6</td>
<td>1.0</td>
</tr>
<tr>
<td>t9</td>
<td>66.2</td>
<td>41.0</td>
<td>0.0</td>
</tr>
<tr>
<td>t10</td>
<td>66.2</td>
<td>40.2</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Cumulative**

<table>
<thead>
<tr>
<th></th>
<th>75.0</th>
<th>56.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intersect</strong></td>
<td>42.6</td>
<td>18.1</td>
</tr>
</tbody>
</table>

44% of executable statements were not exercised in acceptance test. They may have been executed in system/unit testing.
<table>
<thead>
<tr>
<th>Procedures</th>
<th>Executed (%)</th>
<th>Executed Statements (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cumulative</td>
<td>80.0</td>
<td>64.9</td>
</tr>
<tr>
<td>Intersection</td>
<td>27.9</td>
<td>10.3</td>
</tr>
</tbody>
</table>

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:

Acceptance test

\[
\begin{array}{c}
.3\\
\end{array}
\begin{array}{c}
55.7\\
\end{array}
\begin{array}{c}
8.4\\
\end{array}
\]

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.