

# **Change and Defect Models and Metrics**

# Change Models and Metrics

---

These models and metrics capture information about modifications to the software system documents

They can be categorized in a variety of ways, usually on a nominal scale

Models are typically presented as distributions of the various classes of changes

Change models and metrics can be used to  
characterize the project

provide insight into the selection of the appropriate processes

provide insight into the product

# Change Models and Metrics

---

## Change Data Classes

Changes can be categorized by

**purpose** e.g., enhancement, adaptive, corrective, preventive

**type** e.g., requirements, specification, design, architecture, planned enhancements, insert/delete debug code, improve clarity, optimize: space or time, feature, enhancement, bug

**cause** e.g., market/external and internal needs

**size** e.g., number of lines of code or number of components affected,

**deposition** e.g., rejected as a change, not relevant, under consideration, being worked on, completed, saved for next enhancement

**level of document changed** e.g., changes back to requirements document

**number of customers affected** e.g., effects certain customer classes

# Change and Defect Models and Metrics

---

## Example Definitions

**Enhancement change:** an addition to the original requirements observable to the user, e.g., adding a new functional capability

**Adaptive change:** modification based upon the environment in which the system lives but does not add new capability observable to the user, e.g., modifying the system to interact with a change to the hardware or operating system

**Corrective change:** a modification made to correct a problem with the existing software solution, e.g., fixing a bug

**Preventive change:** a modification that will make the system easier to modify in the future, e.g., restructuring the modules

# Change and Defect Models and Metrics

---

## Issues associated with nominal scale data

Changes are usually classified on a nominal scale, based upon a classification scheme

It should be easy to classify an object into its nominal category

The categories need to be well defined

The categories need to be orthogonal

The categories need to have intuitive meaning for the classifier

# Change Models and Metrics

---

## Sample Change Metrics

number of enhancements per month

number of changes per line of code

number of changes during requirements

number of changes generated by the user vs. internal request

number of changes rejected/ total number of changes

Change Report history profile

# Defect Models and Metrics

---

## Defect Definitions

### Errors

Defects in the human thought process made while trying to understand given information, to solve problems, or to use methods and tools

### Faults

Concrete manifestations of errors within the software

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

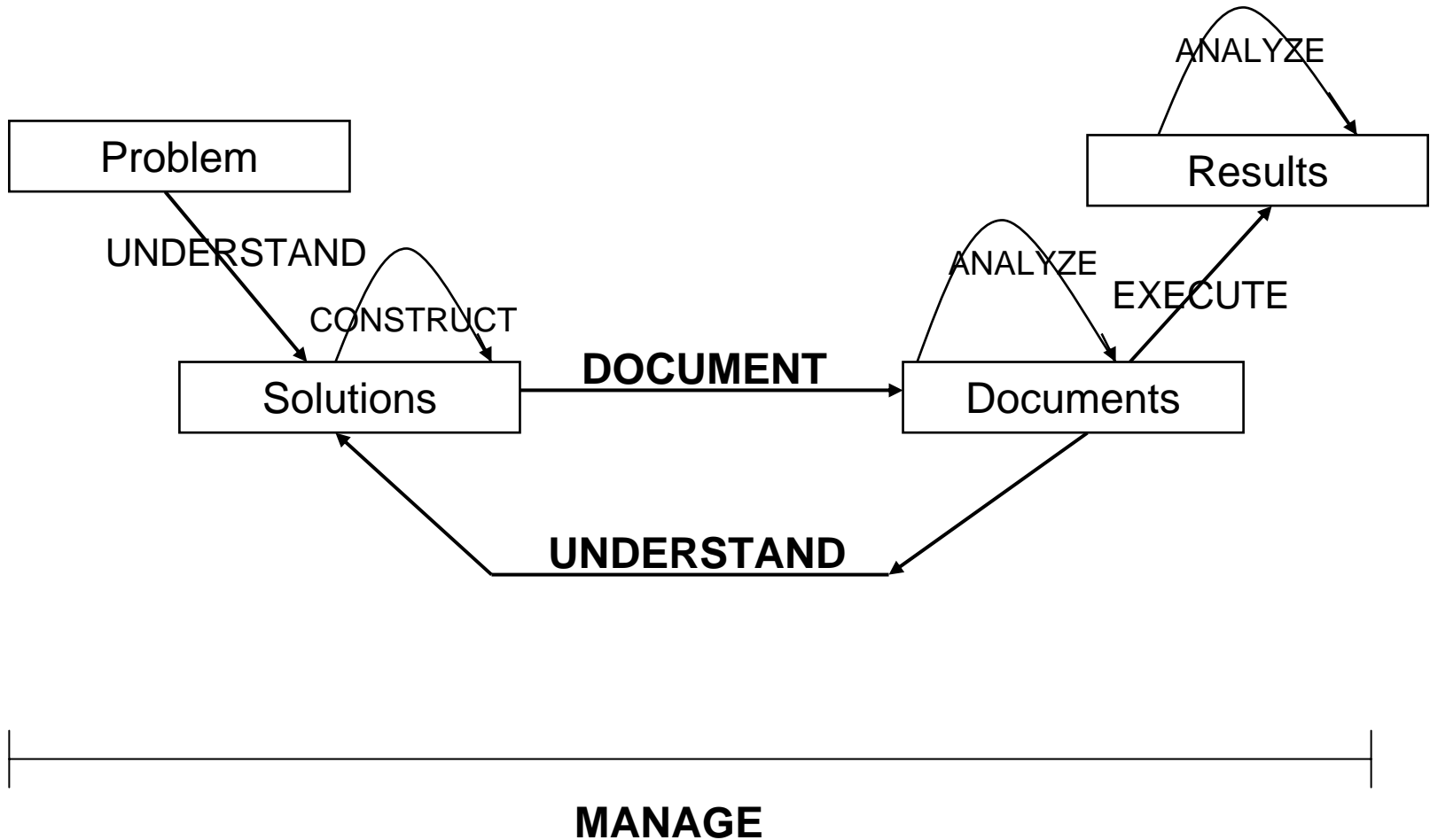
### Failures

Departures of the operational software system behavior from user expected requirements

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

# Building Models

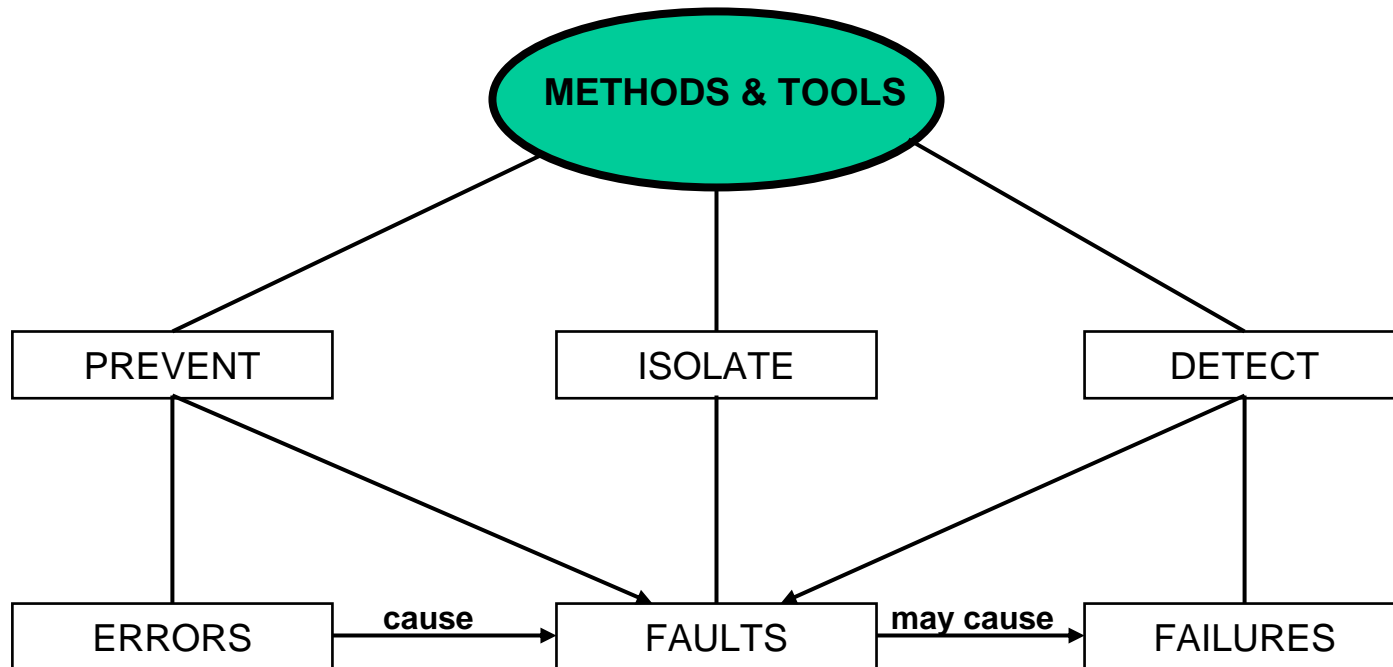
## Methods and Tools Dealing with Software Defects



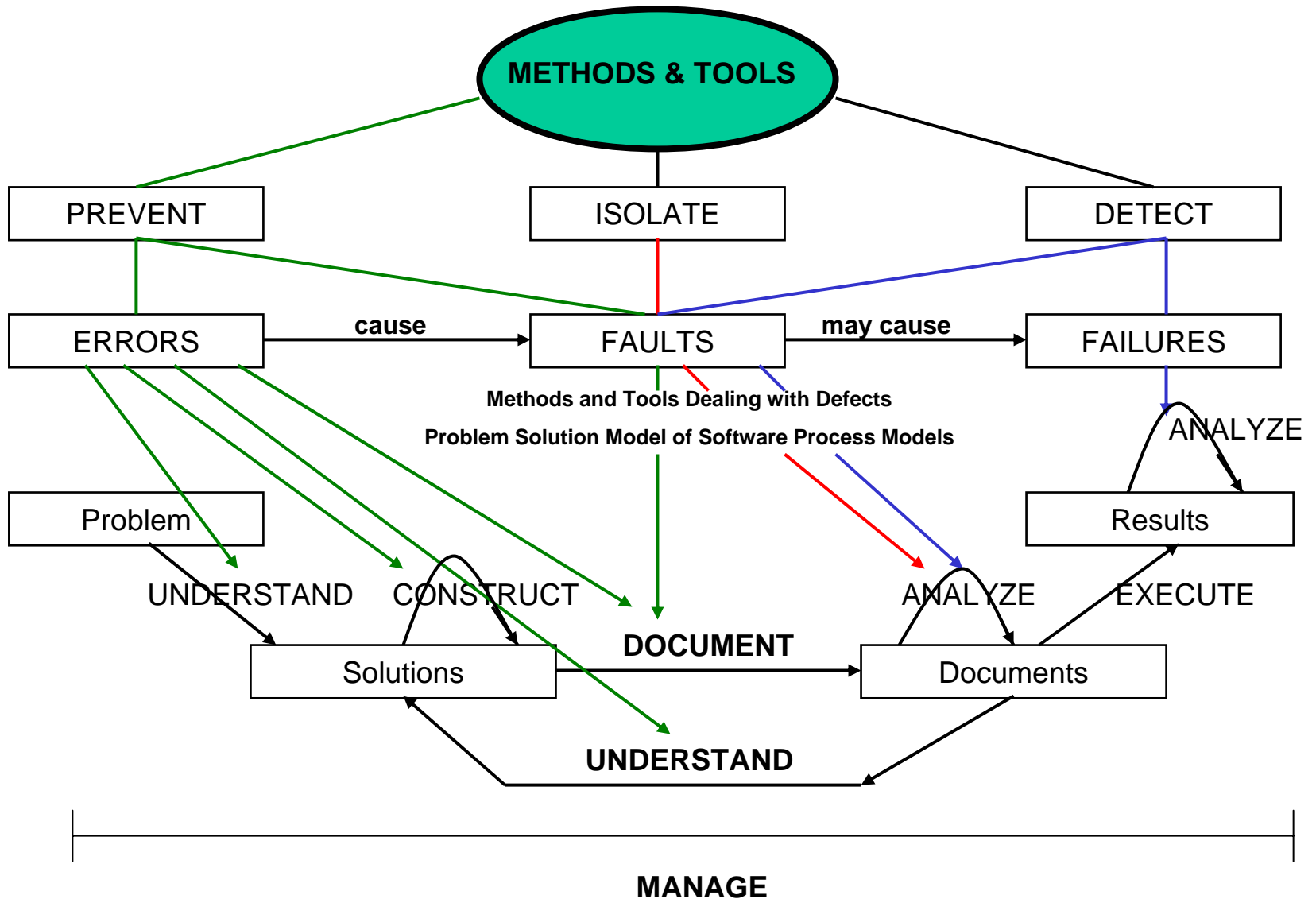
# Building Models

---

## Methods and Tools Dealing with Defects



# Combining Models



# Defect Models and Metrics

---

## Error Data Classes

**Error origin** - the phase or activity in which the misunderstanding took place.

Example subclasses: requirements, specification, design, code, unit test, system test, acceptance test, maintenance

**Error domain** - the object or domain of the misunderstanding.

Example subclasses: application, problem-solution, notation <semantics, syntax>, environment, information management, clerical

**Document error type** - the form of misunderstanding contained in a document.

Example subclasses: ambiguity, omission, inconsistency, incorrect fact, wrong section

**Change effect** - the number of modules changed to fix an error

# Defect Models and Metrics

---

## Fault Data Classes

**Fault detection time** - the phase or activity in which the fault was detected.

Example subclasses: requirements, specification, design, code, unit test, system test, acceptance test, maintenance

**Fault Density** - number of faults per KLOC

**Effort to Isolate/Fix** - time taken to isolate or fix a fault usually in time intervals

Example subclasses: 1 hour or less, 1 hour to 1 day, 1 day to 3 days, more than 3 days

**Omission/commission** - where omission is neglecting to include some entity and commission is the inclusion of some incorrect executable statement or fact

**Algorithmic fault** - the problem with the algorithm

Example subclasses: control flow, interface, data <definition, initialization, use>.

# Defect Models and Metrics

---

## Failure Data Classes

**Failure detection time** - the phase or activity in which the failure was detected

Example subclasses: unit test, system test, acceptance test, operation

**System Severity** - the level of effect the failure has on the system

Example subclasses: operation stops completely, operation is significantly impacted, prevents full use of features but can be compensated, minor or cosmetic

**Customer Impact** - the level of effect the failure has on the customer

Example subclasses: usually similar to the subclasses for system severity but filled out from the customer perspective so the same failures may be categorized differently because of subjective implications and customer satisfaction issues

# Defect Models and Metrics

---

## Sample Defect Metrics

Number of faults per line of code

Number of faults discovered during system test, acceptance test and one month, six months, one year after system release

Ratio of faults in system test on this project to faults found after system test

Number of severity 1 failures that are caused by faults of omission

Percent of failures found during system test

Percent of interface faults found by code reading

# Building Defect Baselines

---

## Error Origin Classification

What are the major problem areas in our projects?

Is there a pattern? I.e., do similar types of problems arise again and again?

Can we build baselines that characterize our errors, faults or failures?

Can we use these baselines to evaluate the effectiveness of a new technology or other form of change in our environment?

Is there a pattern associated with the defects that will support prediction?

# Building Defect Baselines

---

## Error Origin Classification

Environment:

NASA/GSFC, SEL

Ground support software for unmanned spacecraft control

50K to 120K source lines of Fortran code

Design through acceptance test

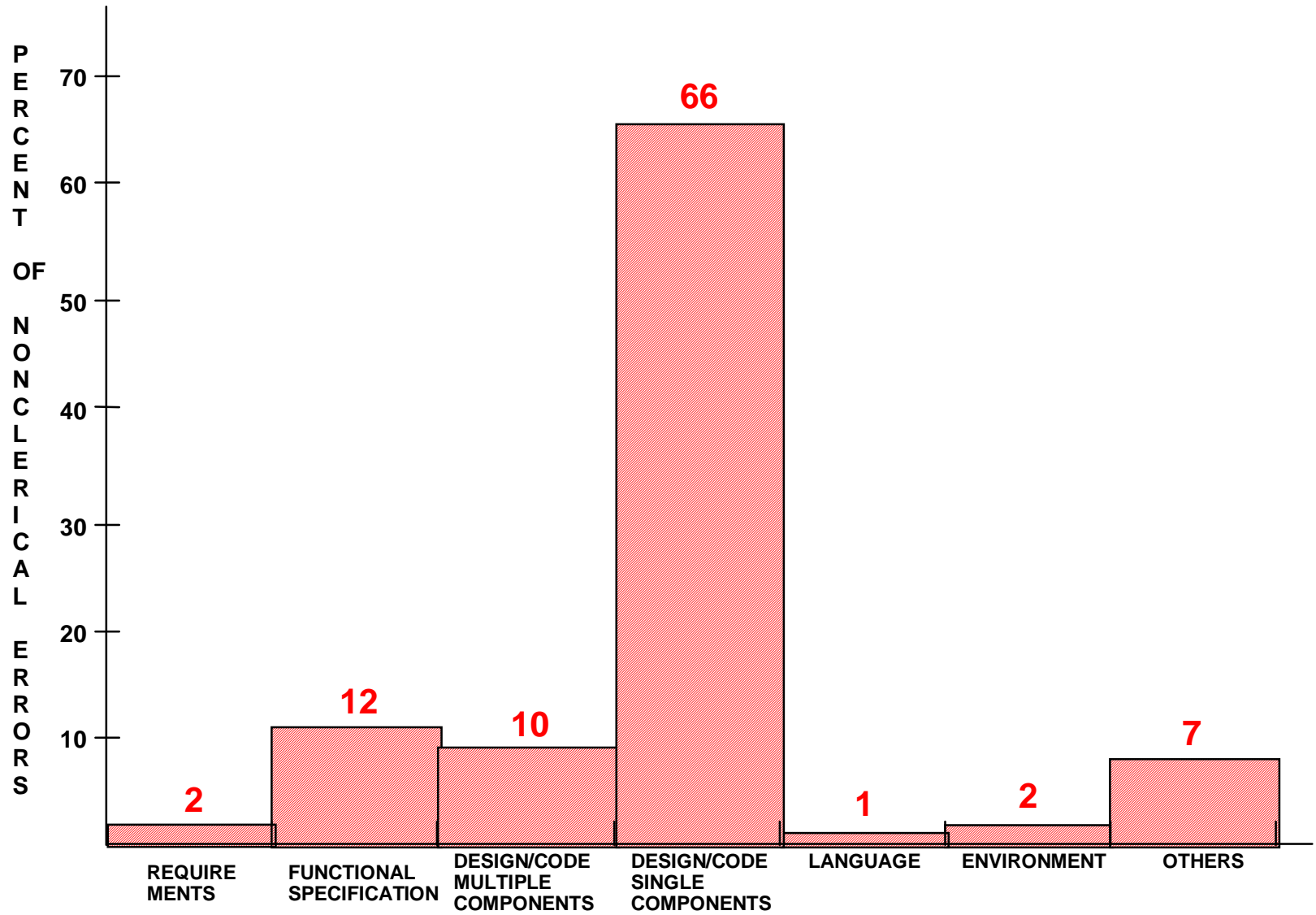
Example Classification

What is the distribution of errors by error origin (i.e., according to the phase in which the misunderstanding took place)

Phases: Requirements, Functional Specification, Design or Coding of Multiple Components, Design or coding of a single component, Language, Environment, Other.

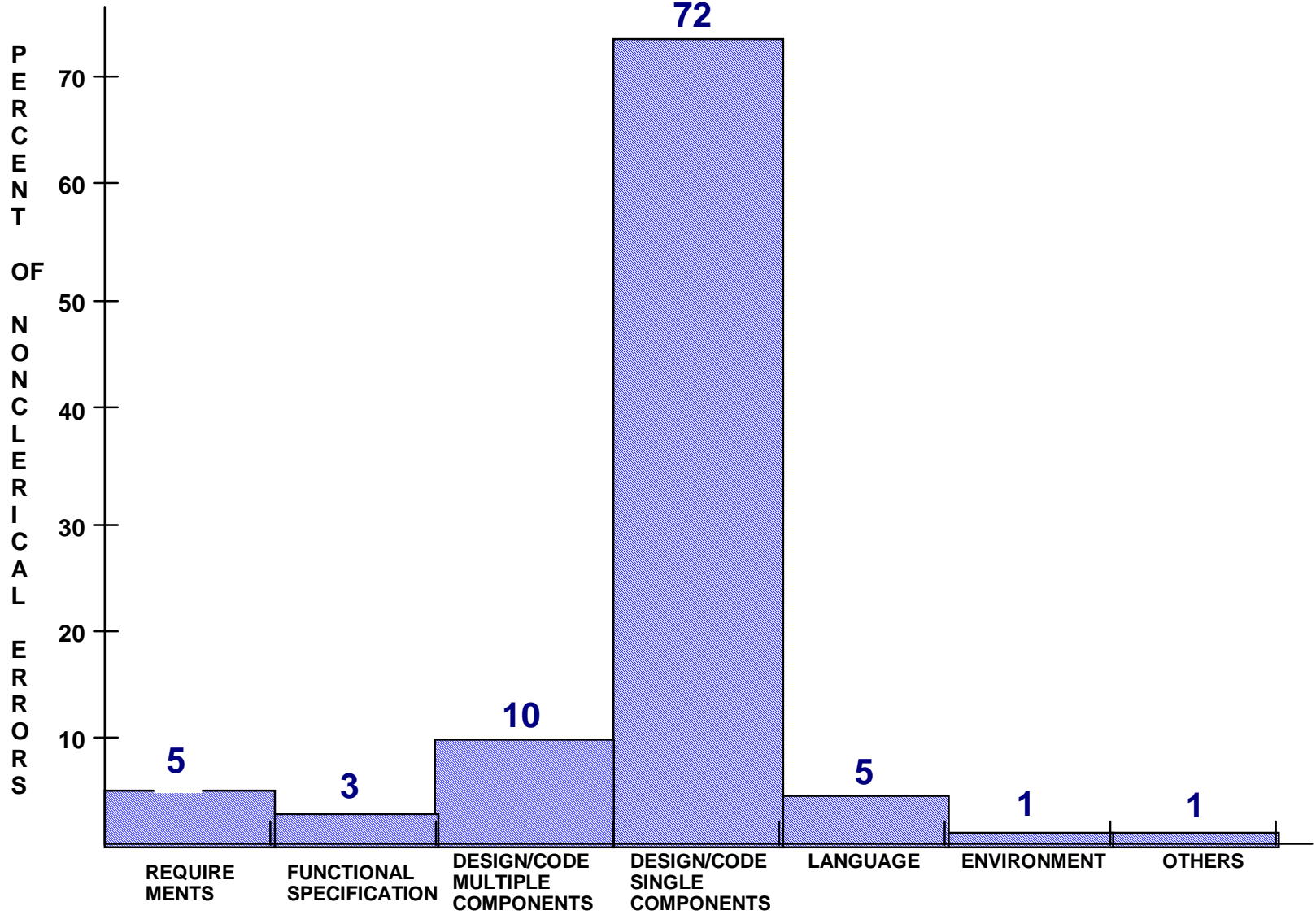
# ERROR

SEL1



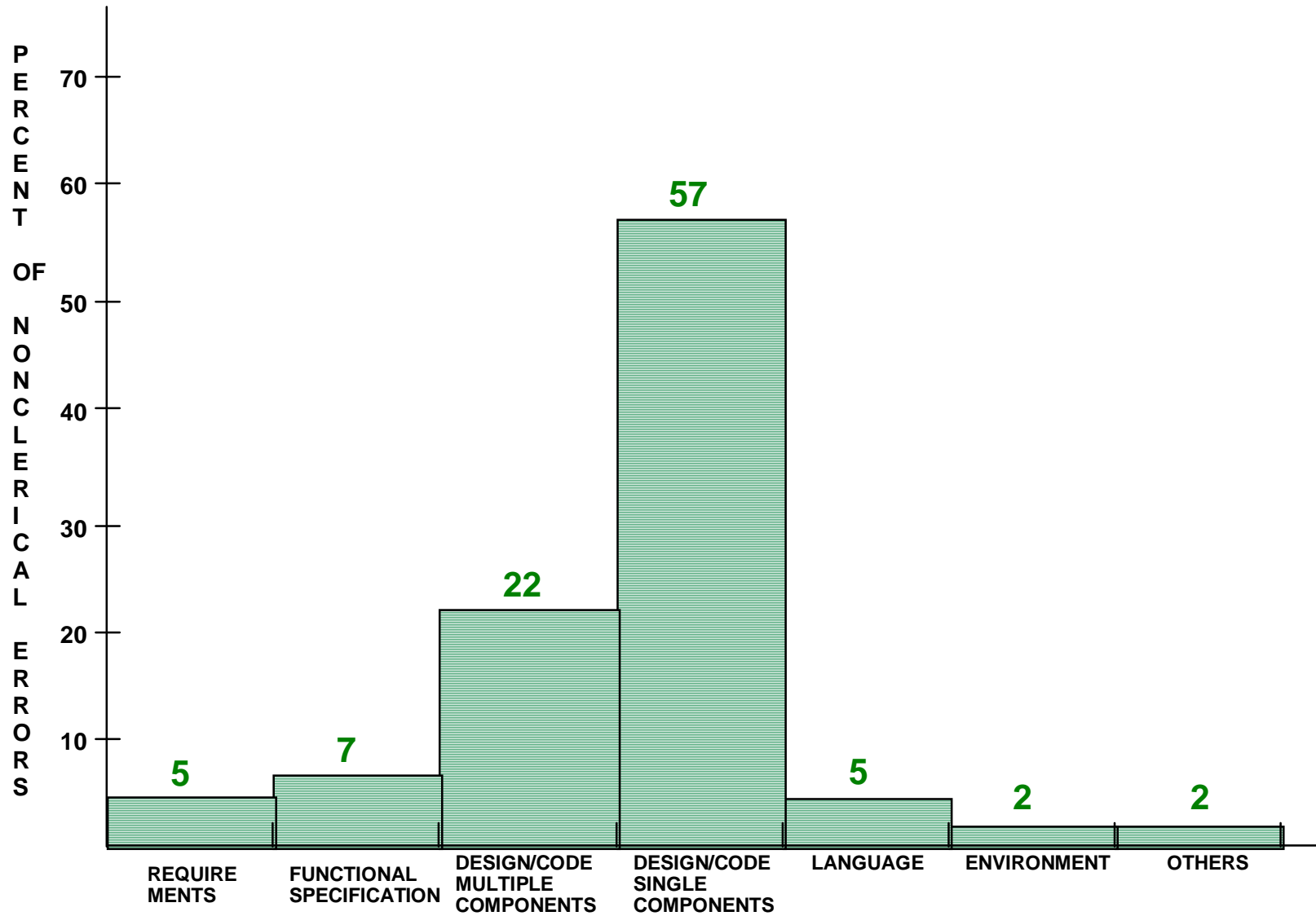
# ERROR

SEL2



# ERROR

SEL3



# Building Defect Baselines

---

## Error Origin Classification

There is a pattern

Not what might have been expected

There were many changes to the requirements and function specification which were classified as changes rather than defects

Several defects were identified and changed before entering the test phase

# Understanding Defect Causes

---

Are some modules more defect prone than others?

What kinds of misunderstandings cause errors?

How many of faults are associated with the problem definition rather than the implementation?

How many of faults are associated with the more than one module?

How many of faults are interface faults?

# Understanding Defect Causes

---

## Environment:

IBM Germany

Operating System Release

~ 500 modules affected by the modification

average size ~360 LOC (480LOC with comments)

432 faults reported

## 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

# Fault Classification

---

## Defect Distribution by Modules

**Number of Defects**

**Number of Modules Affected**

371 (85%)

1

50

2

6

3

3

4

1

5

1

8

Number of modules affected means the number of modules that had to be changed

# Fault Classification

---

## Defect Distribution by Modules

Number of Defects per Module

Number of Modules

Number of Defects/Modules

112  
36  
15  
11  
8  
2  
4  
5  
3  
2  
1  
1  
1  
1

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
14  
15  
19  
28

# Fault Classification

---

## **Problem Specific Defects 46%**

e.g., Machine Configuration and Architecture, Dynamic Behavior and Communication between Processes; Functions Offered, Output Listings and Formats, Diagnostics, Performance

## **Implementation Specific Defects 38%**

e.g., Initialization (of Fields and Areas), Addressability (in the sense of the assembler), Reference to Names, Counting and Calculating, Tasks and Comparisons, Estimation of Range Limits (addresses and parameters), Placing of instructions within a module (bad fixes)

## **Textual Defects 16%**

e.g., Spelling in messages and commentaries, Missing commentaries or flowcharts, (standards), incompatible status of macros or modules, (integration defects), Not classifiable

# Fault Classification

---

## Conclusions

Interface between modules not a major source of defects, when interface is defined in terms of the number of modules changed

Some modules are much more fault prone than others.

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)

# Building Defect Baselines

## Error Origin Classification

---

### Environment:

- NASA/GSFC, SEL
- General purpose program for Satellite Planning Studies
- 90K source lines of Fortran code
- A new system with the requirements continually changing and evolving over time

### Example Classification

- What is the distribution of errors by error origin (i.e., according to the phase in which the misunderstanding took place)

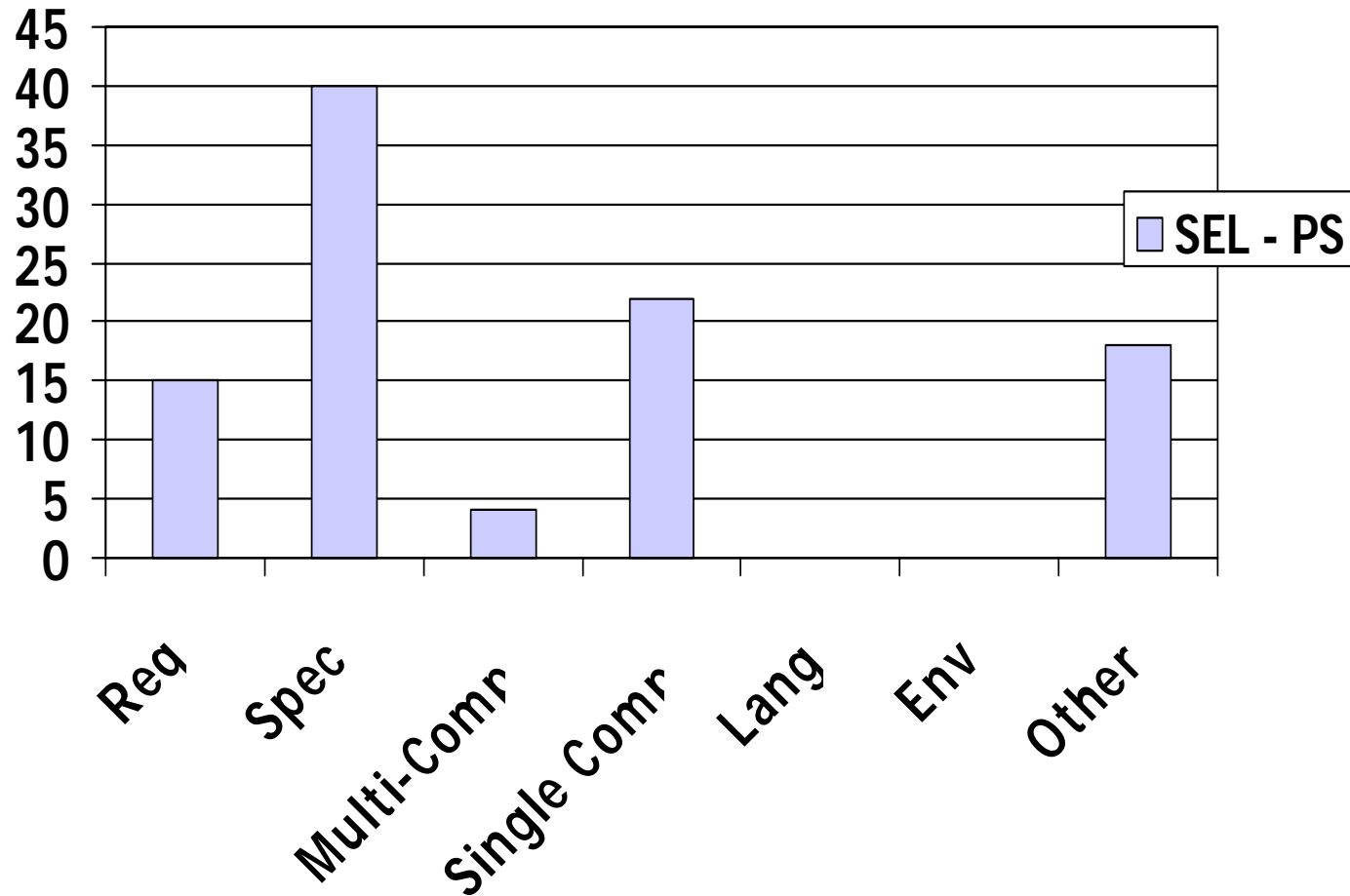
### Phases:

- Requirements, Functional Specification, Design, Coding, Evolution over 3 years.

# Error Origin

## SEL Planning Studies System

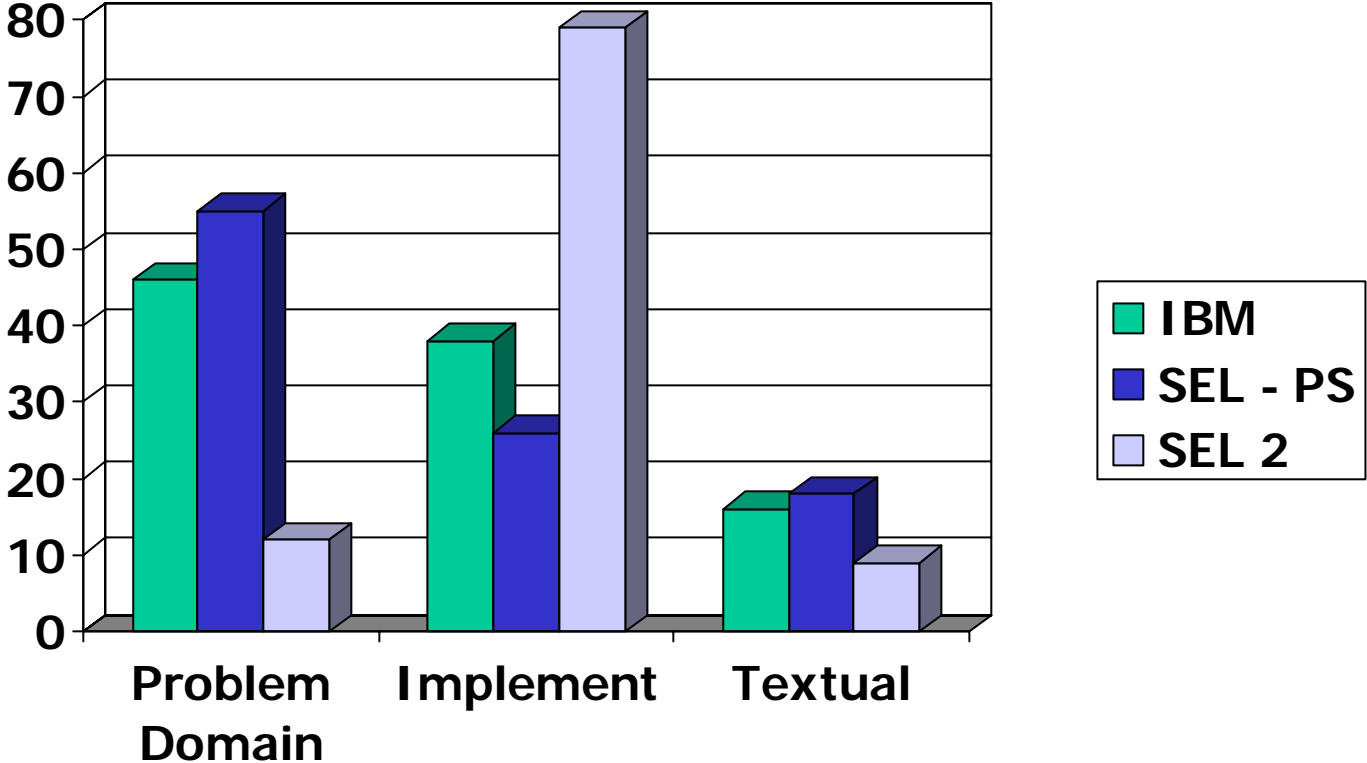
---



# Error Origin

## Old vs. New Application

---



# Interface Fault Definitions

---

## Number of Components Changed:

- It is an interface error if more than one component (module) must be changed to fix the problem
- Implementation interface error

## Number of Components Examined

- It is an interface error if more than one component (module) had to be examined in order to design the change
- Design interface error

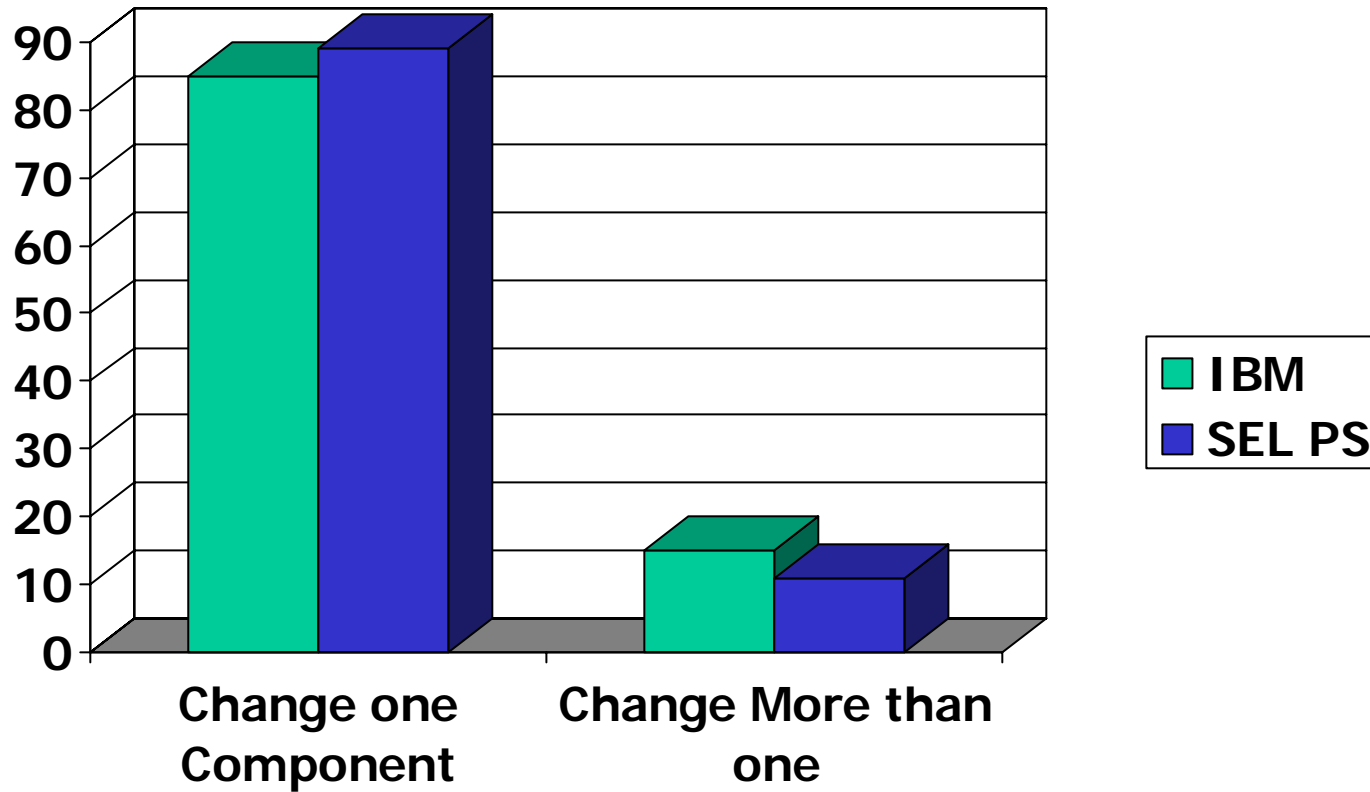
## Classification:

- What is the distribution of interface errors

# Interface Faults

## Number of Components Changed

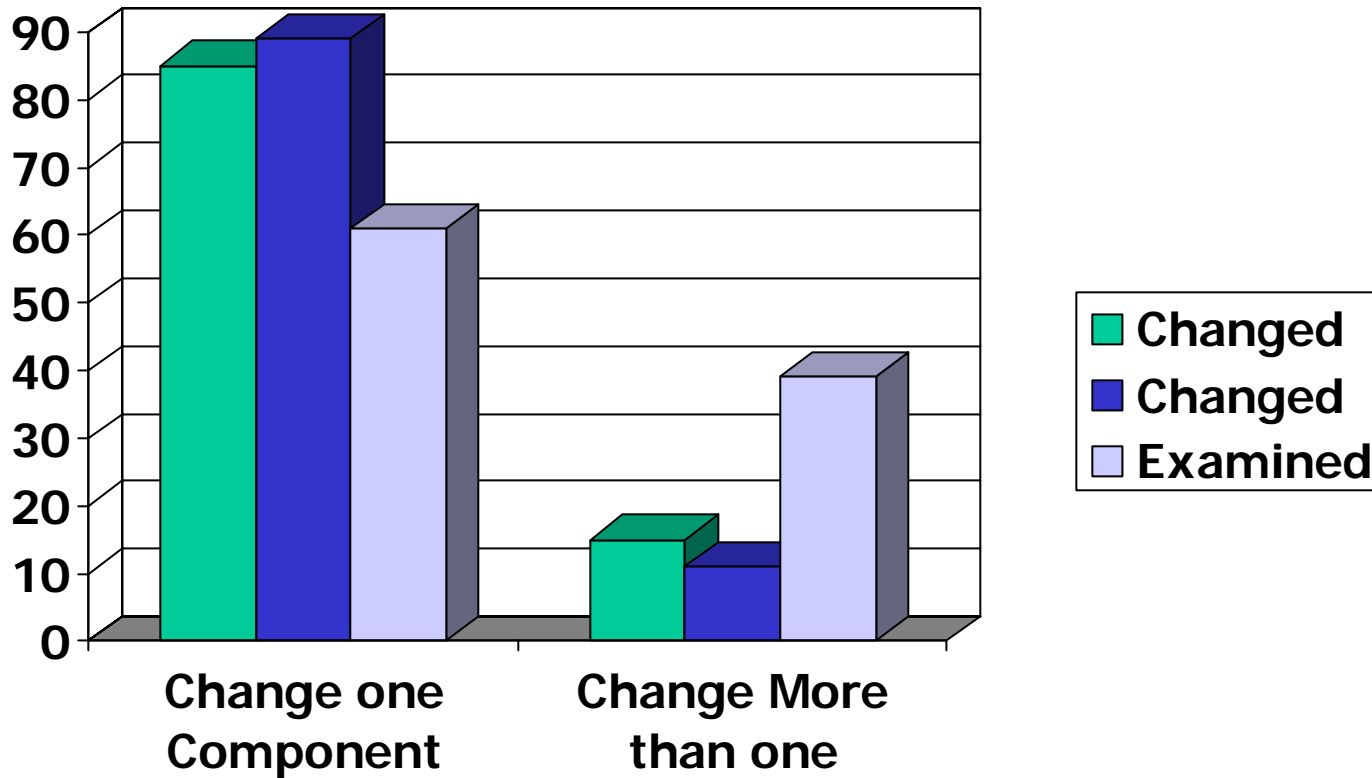
---



# Interface Faults

## Components Changed vs. Examined

---



# Change and Defect Models and Metrics

---

## Issues

What is a change?

What is a defect (error, fault, failure)?

What do you count? When do you start counting?

How do you identify them? How do you compare them from different populations? What are false positives?

What is a change vs. a defect?

How do you classify them? What is the environment? What kinds of meta data do you need to consider? What is the politics involved in the collection process, feedback process?

How do we analyze the data? How do we use it to evaluate? How do we build predictive models? What can we do with this data? How do we understand it outside of the context in which it was collected?

# Models and Metrics

---

## Issues

To use defect data, we need:

- Sound definition of terms

- A model that provides insight into the object, its use or environment

- The model should generate some associated measurement

- The measurement scale and value set need to be well defined

- Concern about the cost and accuracy of the data collected

**Defect data is defect prone**

# References

---

V. Basili and H.D. Rombach, Tailoring the Software Process to Project Goals and Environments, 9th International Conference on Software Eng., Monterey, CA, March 1987.

D. Weiss and V. Basili, Evaluating Software Development by Analysis of Changes: The Data from the Software Engineering Laboratory, IEEE Transactions on Software Engineering, pp. 157-168. February 1985.

Endres, A.: An analysis of Errors and their Causes in System Programs, Proceedings of the International Conference on Reliable Software, pp. 327-336, April 1975, Los Angeles, CA.

V. Basili and B. Perricone, Software Errors and Complexity: An Empirical Investigation, Communication of the ACM, vol. 27, #1, pp. 42-52, January 1984.

# Orthogonal Defect Classification

---

A *function* error is one that affects significant capability, end-user interfaces, product interfaces, interface with hardware architecture, or global datastructure(s) and should require a formal design change.

An *assignment* error indicates a few lines of code, such as the initialization of control blocks or data structure.

*Interface* corresponds to errors in interacting with other components, modules or device drivers via macros, call statements, control blocks or parameter lists.

*Checking* addresses program logic which has failed to properly validate data and values before they are used.

*Timing/serialization* errors are those which are corrected by improved management of shared and real-time resources.

*Build/package/merge* describe errors that occur due to mistakes in library systems, management of changes, or version control.

*Documentation* errors can affect both publications and maintenance notes.

*Algorithm* errors include efficiency or correctness problems that affect the task and can be fixed by (re)-implementing an algorithm or local data structure without the need for requesting a design change.