ARS, North America 2008

Reno, Nevada USA

Track 1, Session 4

Practical Software Failure Analysis

Nematollah Bidokhti
Cisco Systems

George de la Fuente
Ops A La Carte

OPS A LA CARTE™
RELIABILITY AND OPERATIONS CONSULTANTS

CISCO

Introduction

- Software failure analysis (FA) is one of the key elements of development process
- After completing this talk, you will
  - Become familiar with the concept of:
    - SW FA process
    - Failure Modes Taxonomy
    - SW FMEA
- What is FA?
- Three Levels of FA?
- Applying FA in System Test
- Find out how it applies to your work place

Drivers

- Reduce defects found in the design & field
- Improve system performance
- Lack of appropriate specification, implementation or verification of SW requirements
- Reduce time to market
- Increase customer satisfaction

Overview

- Software failure analysis and root cause analysis can help determine the **weaknesses** of the development processes and **reduce the defect density**
- The findings of failure and root cause analysis can be shared amongst all development teams through best practices documents, **failure modes taxonomy** or design guidelines and used to drive process improvements
- The results of failure analysis should be incorporated in the **development training** and **design processes**

Vocabulary

- FA – Failure Analysis
- CFD – Customer Found Defects
- RMA – Return Material Authorization
- TAC – Technical Assistant Center

Differences Between HW & SW FA

Hardware

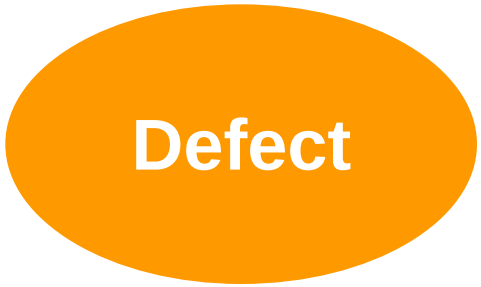
1. Customer Found Defect (CFD)
2. Call in TAC & get RMA
3. HW sent for repair
4. Test performed and failed component identified
5. Component sent for FA
6. FA results communicated to the design team
7. Enhanced design

Software

1. Customer Found Defects (CFD)
2. Call in TAC & SW case opened
3. Case reviewed
4. Assigned to SW designer
5. Based on severity, resolution provided
6. Root cause found, category assigned, fixed and verified
7. Escape analysis
8. SW release updated

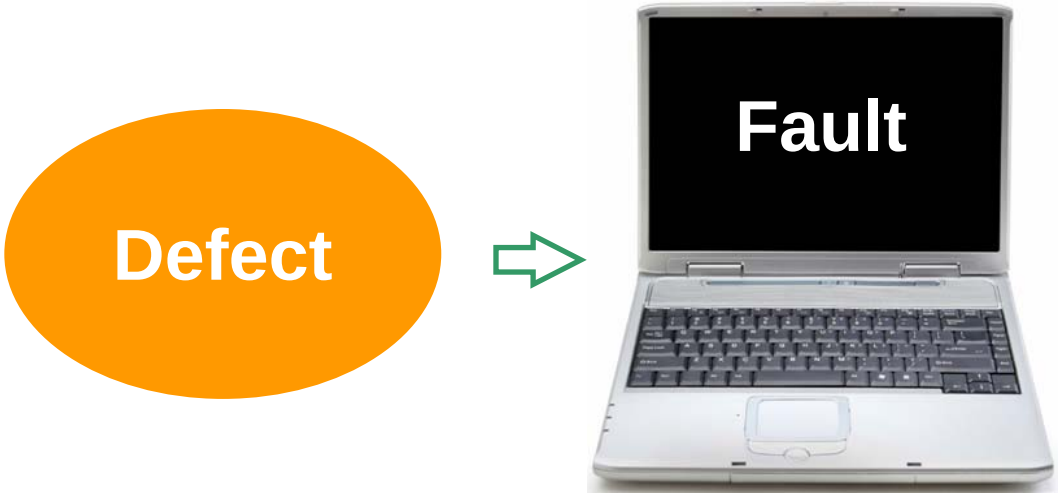
Defect

- An error, flaw or mistake in software requirements, design or source code that prevent it from behaving as intended



Fault

- Any defect that only occur by executing the code
  - It could be customer visible or not
    - Packet drop threshold
    - Congestion

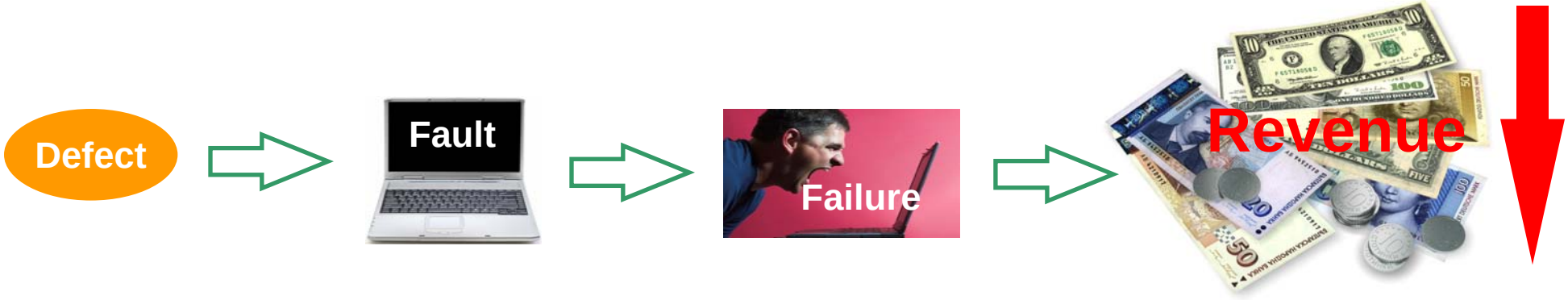


Failure

- Any fault that cause software behavior not to meet its specific requirements
  - Not all failures result in system outages

Impact of Defects and Failures

- There are 3 types of run-time defects
  - Defects that are never executed (so they don't trigger faults)
  - Defects that are executed and trigger faults that do NOT result in failures
  - Defects that are executed and trigger faults that result in failures



What is Failure Analysis (FA)?

- Definition:

  The process of **collection and analysis of data** to determine the **cause of failures** and how to **prevent it from recurring**

- Based on this definition, the key areas of any FA process are:
  - Gather defect or failure data
  - Useful and practical way to determine the root cause of failure
  - Adjust your process to improve the next time

What is the Rationale for FA?

- Used as a vital tool in the electronics (Hardware & Software) industry to develop and improve products
- Enables engineering organizations to determine the weaknesses in their development processes in order to make necessary process improvement changes

The Effects of Software FAs

1. Increase software reliability
2. Meet and exceed Customer expectations
3. Focus on Customer impact
   - What caused the defect
   - Where the defect was introduced
   - Type of defect
4. Enhance system software based on field experience

Three (3) levels of FA

1) Traditional FA applied after the system test phase
2) FA applied at end of each development and test phase
3) FA applied at the beginning and end of each development and test phase

Level 1 FA

- Rationale
  - Determine the failure areas that were most problematic and not detected and addressed until the end of the process, i.e., the system test phase
- When
  - Applied at the end of the software release development cycle, i.e., the system testing phase

Level 1 FA, continued

- Process
  - Extract the failure data from the bug tracking system at end of system test, i.e., bugs that were logged by the testers
  - Classify each logged failure against a reference failure mode taxonomy
  - Determine what upstream development verification process to change in the next release cycle in order to prevent the most problematic failure modes from recurring or reduce the magnitude of their reoccurrence during the next system test phase
    - Apply a pareto division of the taxonomy data in order to focus only on the most problematic categories of the taxonomy
    - For each failure category identified, determine which development phase verification step to target for a process improvement in order to detect these types of failure modes earlier in the development process

Software Failure Modes Taxonomy

- Is the classification of software failure modes by various categories
- Following are some of the SW taxonomy from Reifer, Ristord and Lutz:
  - Large software projects
    - Computational
    - Logic
    - Data I/O
    - Data handling
    - Interface
    - Data definition
    - Database

Software Failure Modes Taxonomy

- General failure modes at processing unit level
  - Operating system stops
  - Program stops with clear message
  - Program stops without clear message
  - The program runs, but produces obviously wrong results
  - The program runs, producing apparently correct but in fact wrong results
- Data or processing of data failure modes
  - Missing data (i.e. lost message)
  - Incorrect data (i.e. inaccurate data)
  - Timing of Data (i.e obsolete data)
  - Extra data (i.e. data overflow)

Level 2 FA

- Rationale
  - Determine the failure areas that were most problematic for each development and test phase, i.e., different types of issues will be more prevalent in each phase
- When
  - Applied after the verification step of each development and test phase

Level 2 FA, continued

- Process
  - Extract the defect or failure data logged during each phase, i.e., defects found during a design or code review, failures found during unit or system testing
  - Classify each defect or failure found against a reference failure mode taxonomy
  - Determine how to change the verification process in the next release cycle in order to prevent the most problematic failure modes from recurring or reduce the magnitude of their reoccurrence in this phase
    - Again, use a pareto approach to find the most problematic categories of the taxonomy
    - For each failure category identified, develop a verification process change that will focus on finding these types of failures, e.g., perspective-based or checklist-based reviews or different targeted testing methodologies

Phase Containment

- Design teams should make every effort to identify and fix problems at the earliest possible point in the development life cycle
- It is very well known that the longer a problem persists, the more costly it will be to eventually correct
- Phase containment measures the quantity of problems escaping the earliest possible review (containment) points

Phase Containment

- The higher the number of escapes, the more likely the project will experience delays, quality problems and/or cost overruns
- The phase containment metric makes a distinction between problems discovered during in-phase reviews and those that have escaped the in-phase review and are found in downstream review points
- Ideally, all design-oriented problems would be found in-phase and none would escape to a future phase
  - Examples of project faults
    - Documentation errors
    - Architecture errors
    - Design errors
    - Coding errors
    - Testing errors

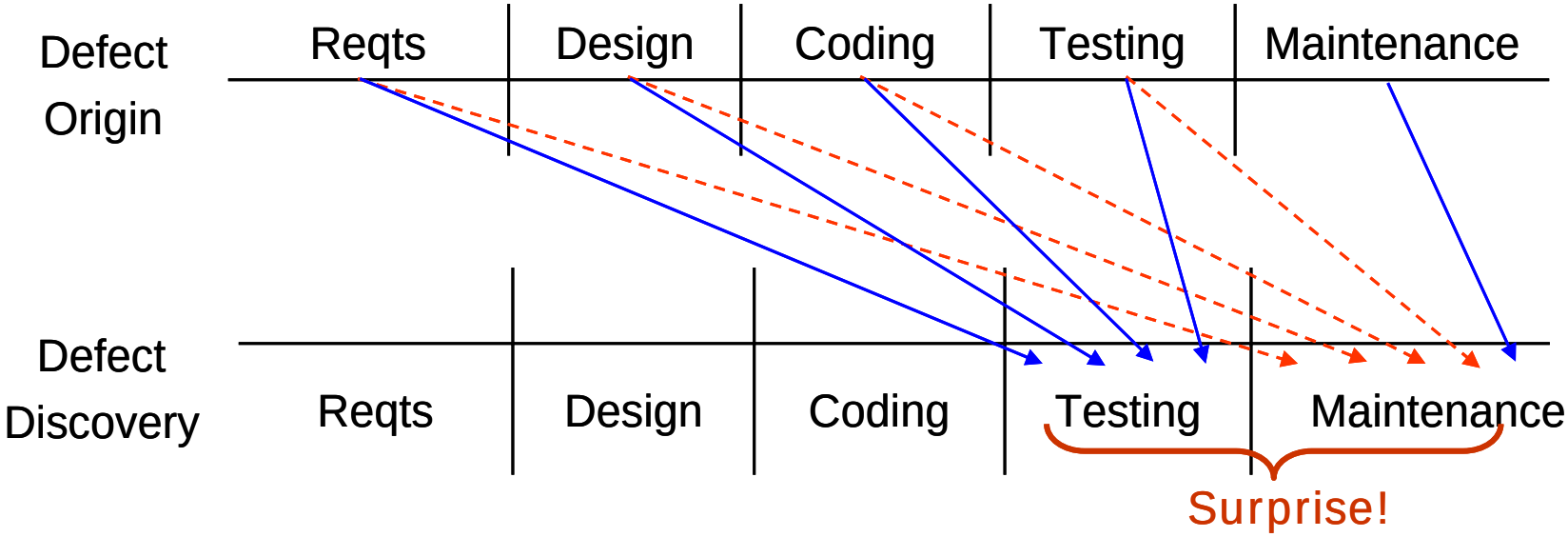
Phase Containment

- A solid project should find high proportion of the total faults as early as possible and well before they impact customers

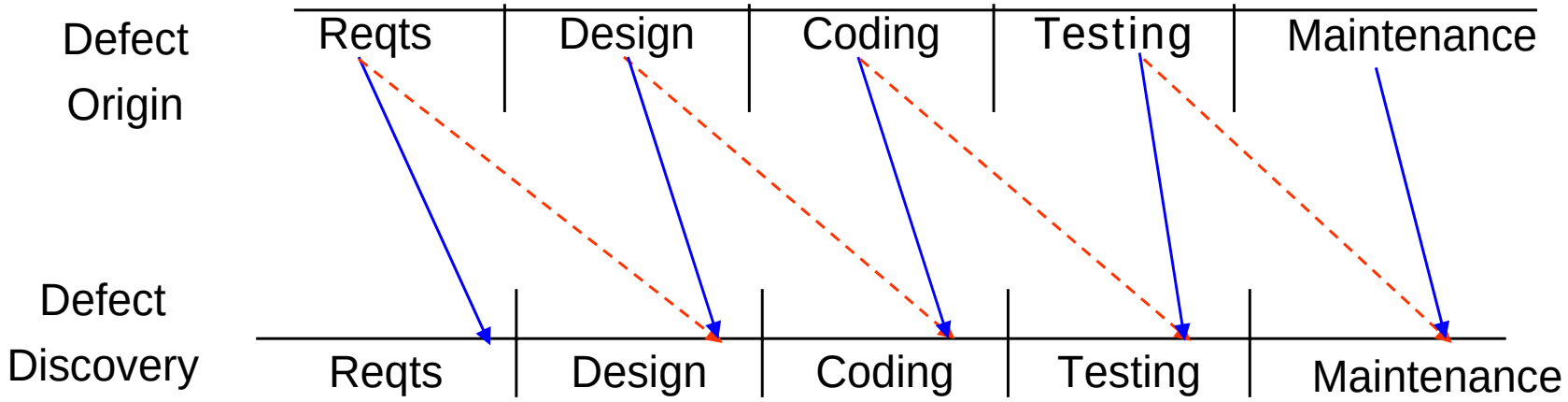
	FS Review	HLD Review	LLD Review	Code Review	Unit Test	Integration Test	System Test	Total Errors	Total Defects	Total Faults	Phase Containment Effeciency
Functional Specification (FS)	**7**	**3**	**2**	**0**	**0**	**0**	**0**	7	5	12	0.58
High Level Design (HLD)		**80**	**30**	**5**	**15**	**11**	**12**	80	73	153	0.52
Low Level Design (LLD)			**109**	**29**	**25**	**14**	**10**	109	78	187	0.58
Code				**89**	**34**	**12**	**2**	89	48	137	0.65
Test Plan					**26**	**10**	**1**	26	11	37	0.70
Faults by Phase	7	83	141	123	100	47	25				

Phase Containment of Defects and Failures

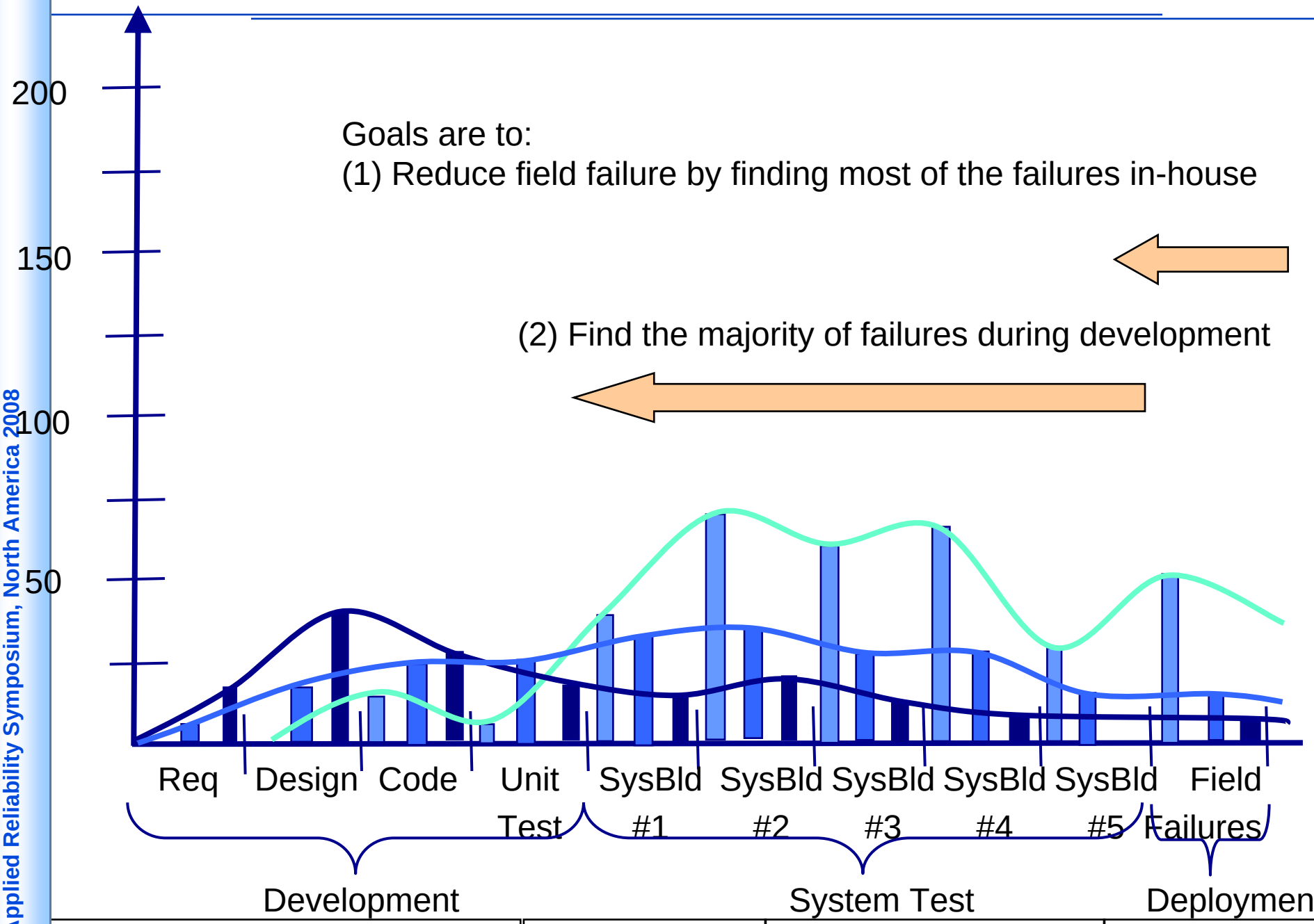
Typical Behavior



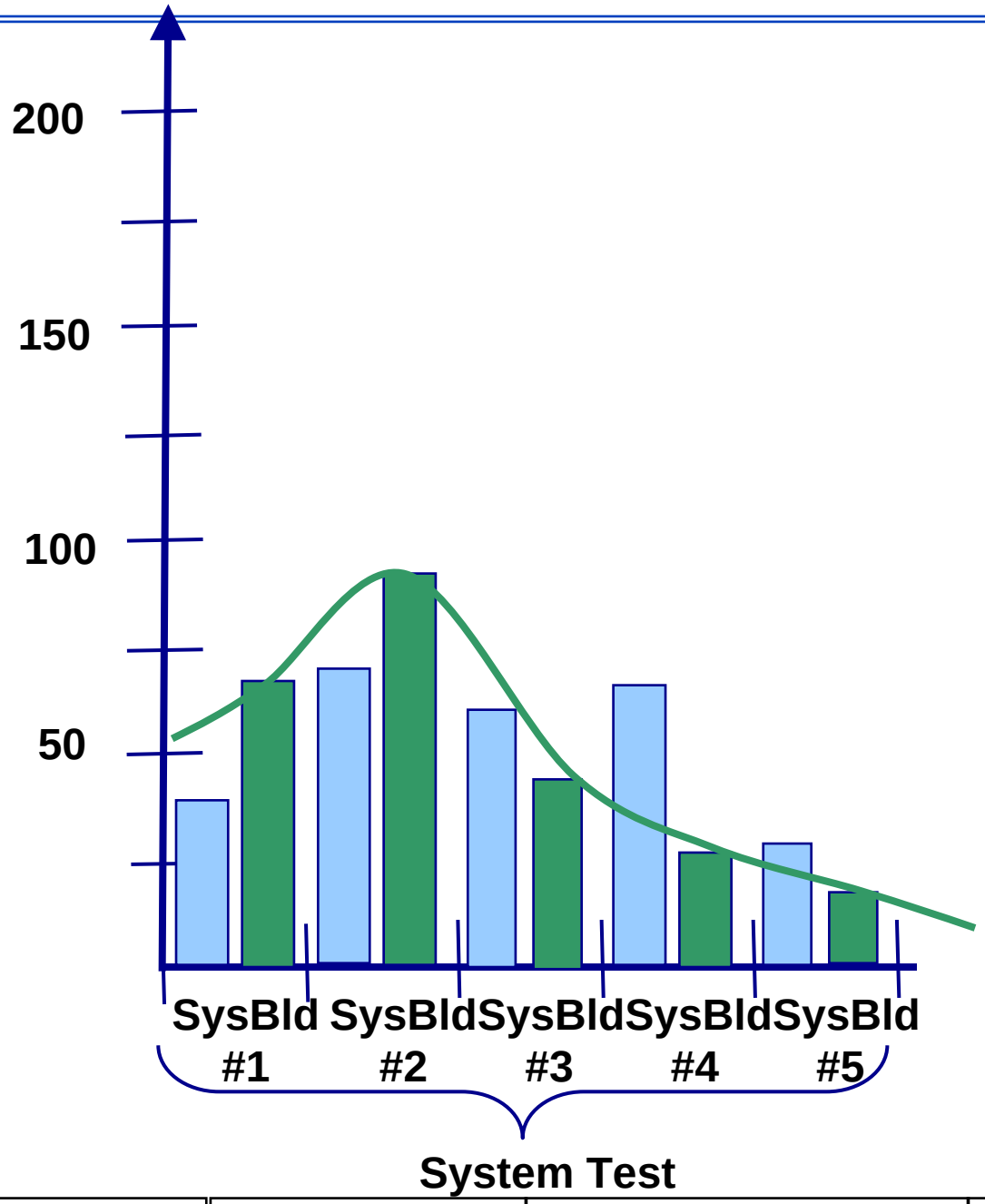
Goal of Phase Containment



DfR Phase Containment Behavior



Typical Defect Reduction Behavior



Where was Defect Found?

1. Identification of activities that discovered the defect
2. Focus the development and system testing on Customer behavior when the defect is encountered
3. Information supplied by defect submitter

Design Review & Code inspection

- Design Conformance
- Understanding Flow
- Backward Compatibility
- Language Dependency

Unit Test, Functional Test, Dev Test

- Basic Function
- Functional Variation
- Functional Interaction

System Test, Performance Test, Interop. Test

- Startup / Restart
- HW Configuration
- SW Configuration
- Error Recovery
- Normal Mode

Defect Origin

1. Show where the defect was originally introduced
2. Identify the development phase where the improvement must take place
3. Capture information supplied by the SW designer / fixer

Defect Origin

- Requirements
- Design
- Code
- Hardware
- Bad Fix

Defect Category

- Standards
- Function
- Error Handling
- Timing
- Interface (Int./Ext.)

Defect Reason

- Missing
- Wrong
- Not Clear

Level 3 FA

- Rationale
  - Augment the Level 2 FA by introducing the FA data early on in order to sensitize the author to the defect or failure issues before the artifact (i.e., document, code or test) is created
- When
  - The additional step for this level is applied at the beginning of each development and test phase

Level 3 FA, continued

- Process
  - At the beginning of each phase, perform a review of the failure mode categories identified in that phase during the previous cycle, i.e., defects found during a design or code review, failures found during unit or system testing.
  - Discuss approaches or methods which the author can use to proactively reduce these types of failure modes during the artifact creation process.
    - In most cases, sensitizing the author ahead of time to the prevalent defect or failure types that are most frequently created will greatly reduce the amount of resulting defects/failures of that type.

FA Level Summary

- The 3 FA levels represent require increasing degrees of process maturity and produce increased reductions in defects and failures during the next cycle
- Continuous execution of FAs allows an organization to develop a comprehensive failure mode taxonomy that targets the problems specific to your teams and development processes

- But, can we get more out of FAs?

Applying FA Results to the Current Release

- Rationale
  - Use partial FA results to make in-situ process phase adjustments during the system test
- When
  - At the mid-point of the system test phase

Applying FA Results to the Current Release

- Process
  - Review the logged failures from the bug tracking system
  - Perform the traditional FA process against this data
    - Using a pareto approach, find the 1-2 most problematic areas in the taxonomy.
    - For each problematic area, map the associated bug fixes to the affected source code files.
    - Use a pareto approach to select the files that contained the most source code changes resulting from fixes.
    - Perform targeted code review analysis of these files looking for more bugs of these types or to determine if major design issues exist (fewer files to focus on for very targeted code reviews).

Two Examples of Root Causes and Remedies

1. **User-interface defect**: There was a way to select (data) peaks by hand for another part of the product, but not for the part being analyzed
   - *Cause*: Features added late; unanticipated use
   - Proposed way to avoid or detect sooner: Walkthrough or review by people other than the local design team
2. **Specifications defect**: Clip function doesn't copy sets of objects
   - *Cause*: Inherited code, neither code nor error message existed. Highly useful feature, added, liked, but never found its way back into specifications or designs.
   - Proposal to avoid or detect sooner: Do written specifications and control creeping features

Software FMEA

- Desired system behavior
  - Communicate to SW designers
  - Perform the SW FMEA
- Even though SFMEA is critical and should be applied to any SW components, there are challenges:
  - There is little historical data available due to verity of components, tools and sw technology
  - Field data are less frequently kept
  - Less experience in categorizing sw failures
  - Experience on the sw are not generally available to due frequent movement among sw engineers among projects
  - SW failures show incorrect behavior and are not perceived as failures
- There are certain standard fields for SW FMEA, but in general each organization must develop a template tailored to their application.

Software Failure Analysis Recommendations

- Must have good organization-wide defect data representation
- Ensure bug tracking system has good query and report capability
- Develop software failure modes taxonomy
- Identify process and product weaknesses
- Change organization design behavior and enhancement based on FA data
- Ensure the training program, development process enhancement and maintenance processes benefit from the FA data
- Maintain FA as part of continuous improvement process

Typical Questions to Ask During FA

- Where was the error made?
- When was the error made?
- What was system reaction?
- What was the HW & SW configuration at the time of failure?
- What was done wrong?
- Why was the particular error made?
- What could have been done to prevent this error?
- If an error could not have been prevented, what detection method could detect it?

Nematollah Bidokhti, Cisco

Nematollah is a technical leader at Cisco Systems. His background includes hardware and software Reliability engineering, system engineering, Fault management, System and network modeling. He has contributed and managed design for reliability activities for military grade, bio-medical, telephony, optical and Data products. He holds a BSEE from Florida Atlantic University.

Contact Info: nbidokht@cisco.com

George de la Fuente, Ops A La Carte

George has 25 years of product development and management experience with embedded systems. His professional software background spans the following industries: telecommunications, networking, gaming, and satellite operations. George has expertise in the following areas: full life-cycle development, rapid prototyping, zero-defect development, sustaining engineering, coding standards, systems testing, release management, software configuration management, project leadership, organizational management and program management George educational background includes an M.S. degree in Computer Science from Santa Clara University and a B.S. degree in Mechanical Engineering from Yale University. George developed the core Software Reliability program, including training modules and services covering software reliability testing, software fault tolerance, software failure analysis, system availability design, and best development practices.

Contact Info: georged@opsalacarte.com