Defect Prevention: A Tester's Role in Process Improvement and reducing the Cost of Poor Quality

Mike Ennis, Senior Test Manager
Accenture

Defect Prevention versus Defect Detection

- Defect Prevention

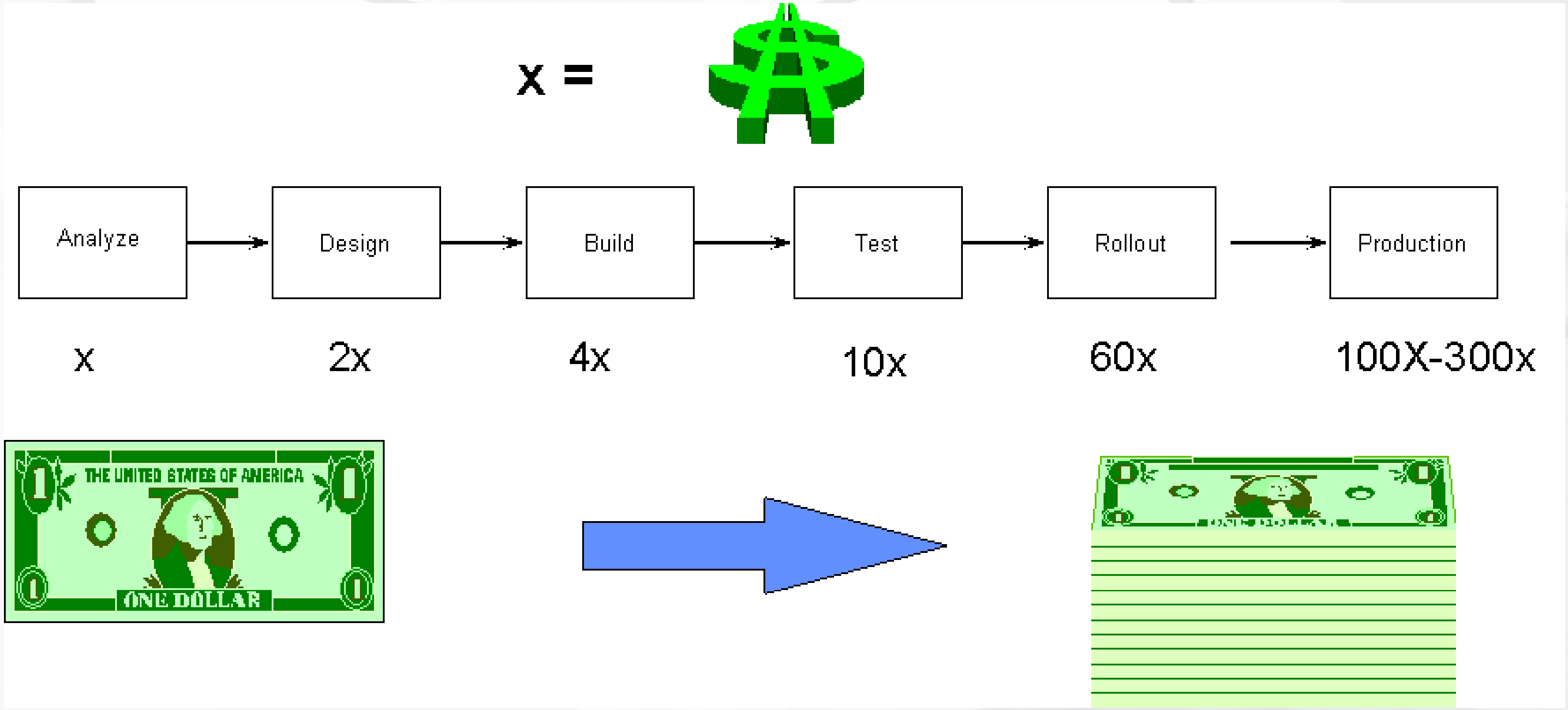
  Techniques that are designed to find defects “before” the artifact has been developed

- Defect Detection

  Techniques that are designed to find defects after the artifact “under test” has been developed

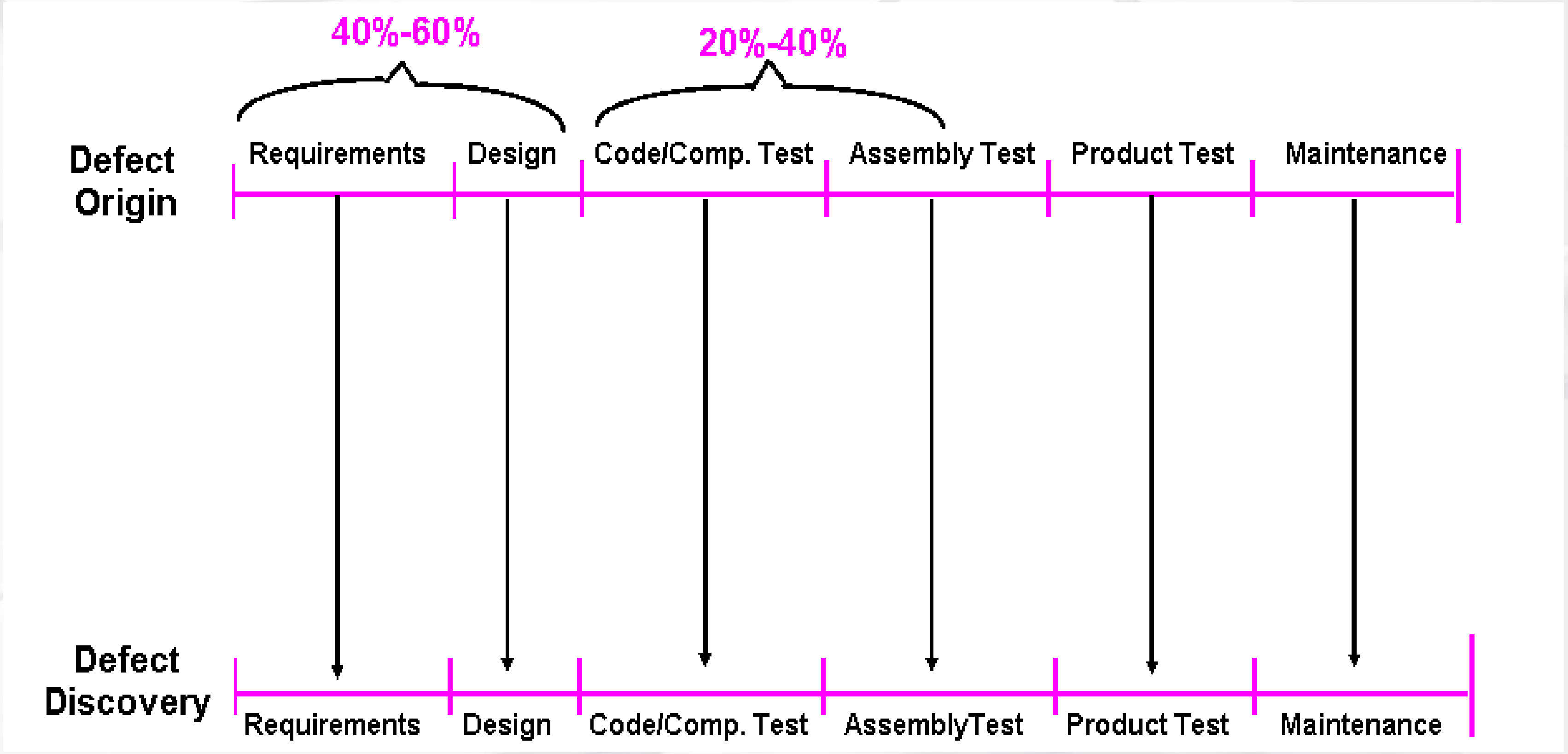
Why is Defect Prevention so important?

The cost factor for discovering and fixing defects in later phases versus early phases can be up to 100:1



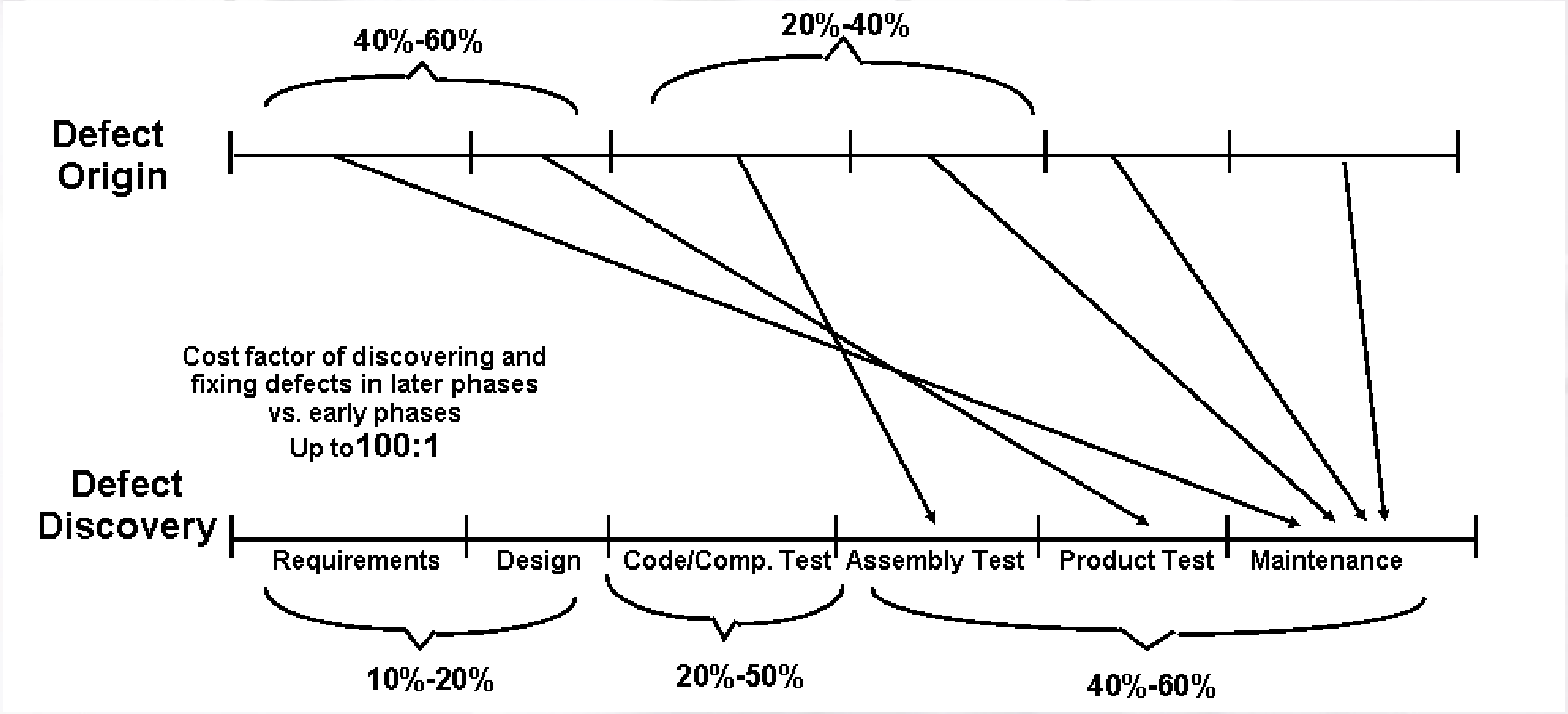
Stage Containment

in a typical IT organization, most defects are injected into a system in its early stages



Stage Containment

in a typical IT organization, most defects are injected into a system in its early stages and fixed in its later stages



Quality Assurance

- Activities that modify the development process to prevent the introduction of flaws
  - Staff function
  - Implements management’s quality policies
  - Responsible for continuous improvement of the software development process
- Proactive approach focused on defect prevention
- Examples:
  - Defining change control procedures
  - Facilitating post-project reviews
  - Analyzing metrics to identify opportunities for process improvement

Quality Control

- Activities within the development process to detect the introduction of flaws
  - Test planning and execution
  - Quality control measures a product against the existence of an attribute
  - Determines whether the product conforms to a standard or procedure (also known as compliance checking).
- Proactive approach focused on defect detection
- Examples:
  - Writing and executing test cases and scripts
  - Participating in verification and validation activities
  - Reporting defects to identify opportunities for process improvement

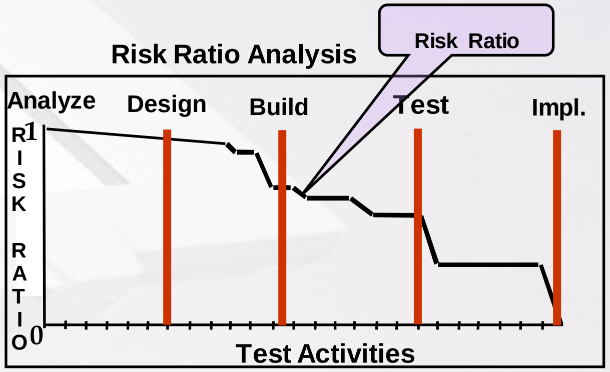
Defect Prevention Techniques

Examples of Defect Prevention:

- Risk Based Testing
- Inspections
- Reviews
- Walkthroughs

Defect Prevention Techniques

Risk Based Testing



Business Value

Risk-based testing is a structured method for reducing testing effort based on component risk. It is a proven practice in testing which relies on strong business and IT participation to deliver real business value. As applied to the development lifecycle, risk management provides a means to focus testing efforts where they will make the greatest difference. Accenture's unique utilization of Risk-Based Testing together with our proprietary assets in this space will make it easier to engage the business units early in the lifecycle, especially for applications where requirements have not been documented thoroughly.

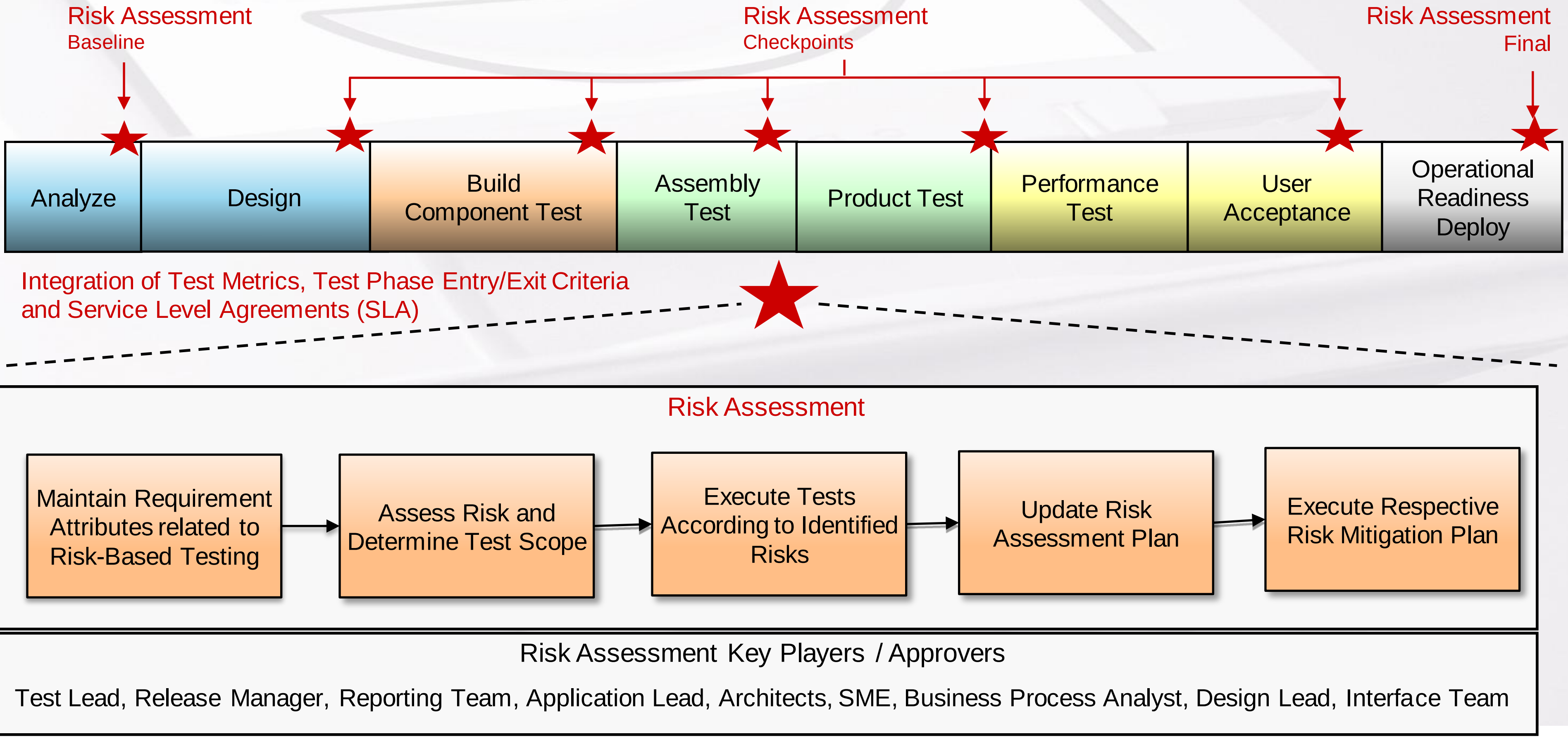
Objectives of Risk-Based Testing

- Minimize threats to the solution delivery objectives.
- Provide a systematic approach for:
  - Evaluating and improving the quality of requirements;
  - Improve both test effectiveness and efficiency;
  - Monitoring and reporting progress in reducing risk;
  - Adjusting risk reduction actions (e.g. test scope) based upon the effectiveness of prior actions or additional knowledge.
  - Gain consensus on high risk requirements

Implementing Risk-Based Testing

The Process within the Solution Life Cycle

The following illustrates the iterative "risk re-assessment" that takes place at each point where a ★ is shown below. Participation from key business, IT, and QA reps is important at each reassessment point.


Risk Assessment
Baseline
Risk Assessment
Checkpoints
Risk Assessment
Final
Analyze
Design
Build Component Test
Assembly Test
Product Test
Performance Test
User Acceptance
Operational Readiness Deploy
Integration of Test Metrics, Test Phase Entry/Exit Criteria and Service Level Agreements (SLA)
Risk Assessment
Maintain Requirement Attributes related to Risk-Based Testing
Assess Risk and Determine Test Scope
Execute Tests According to Identified Risks
Update Risk Assessment Plan
Execute Respective Risk Mitigation Plan
Risk Assessment Key Players / Approvers
Test Lead, Release Manager, Reporting Team, Application Lead, Architects, SME, Business Process Analyst, Design Lead, Interface Team

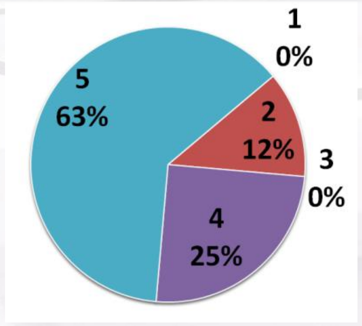
Risk Assessment Analysis

Risk Factor Distribution

It is important to understand the primary factors driving the assessed risk level.

Impact

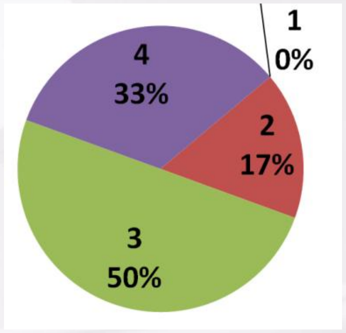
Quantifies the "true" impact of each risk, also referred to as the 'level of damage' potentially caused by the change. This is an estimate of the overall scale of impact if a defect related to the change were found in production.



5 = Critical …
1 = Marginal

Probability

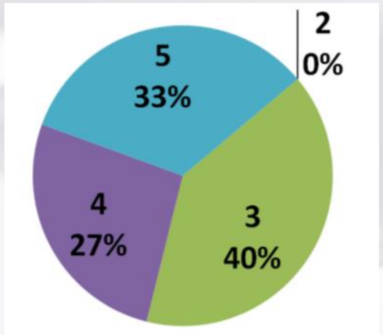
This is an assessment of the likelihood of an error being delivered into production at a point in time.



5 = Very Likely …
1 = Unlikely

Level of Control

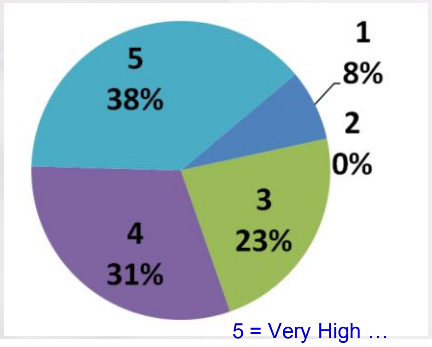
Indicates the relative control which can be exerted on the probability of the risk occurring in production.



5 = Total Control …
1 = No Control

Confidence Level

Indicates the Confidence level of the Risk evaluated. In a typical situation, as we understand and get more clarity on requirements confidence level increases.



5 = Very High …
1 = Very Low

Defect Prevention Techniques

Inspections

- Create an Inspection Plan
- Inspections should raise the quality of an application
- Inspections should find faults with requirements, test plan documents, design documents.
- Similar to a Peer Review

Defect Prevention Techniques

Reviews

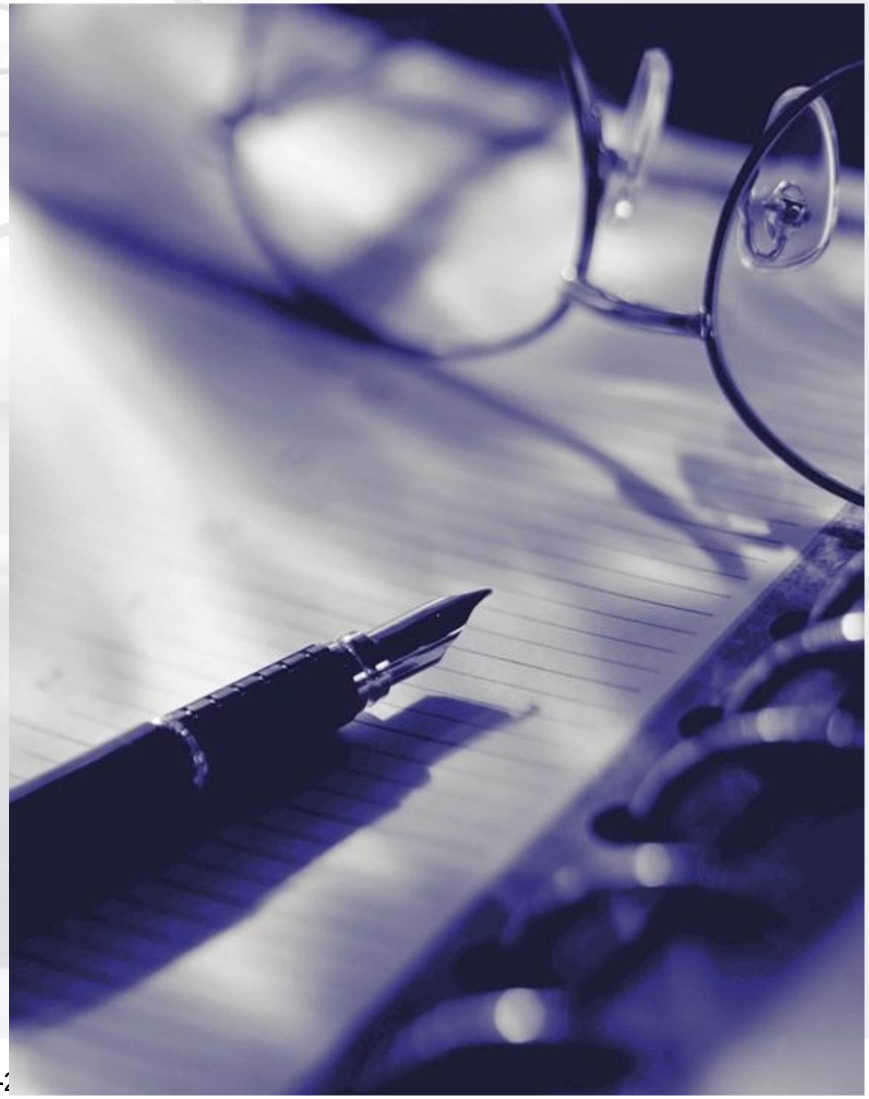
- Establish a Peer Review Process
- Identifies issues/risks in Test Planning
- Types of Reviews:

  -Formal

  -Informal

  -Facilitated

  -Unfacilitated



Defect Prevention

Walkthroughs

- No Formal Plan
- Peer/Lead Checks
- Typically done in a group setting while walking through a test script or document
- Feedback given by attendees during walkthrough

Defect Prevention – Type of Defects Found

Work product elements that are:	
	Missing
	Inconsistent or mismatched (such as interface specifications)
	Non-conforming (fail to follow processes)
	Incorrect (deviate from requirements and rules or standards)
	Unclear
	Ambiguous
	Unnecessary
	Un-testable

Defect Prevention – What Can Be Reviewed?

- Specifications
  - Requirements
  - Design
- Memos
- Proposals
- Plans
- Policies
- Procedures
- Brochures
- Statements of work
- Contracts

Defect Prevention – Value of Reviews

- Provide early quality assessments (verification) of the work products
  - Detecting dependencies and inconsistencies in software models, such as links
  - Improved maintainability of code and design
- Colleagues will see what you can't
- Catch bugs earlier
  - It's cheaper to prevent a defect than to repair one later in the life cycle
- Identification of defects not easily found by dynamic testing

Defect Prevention – Value of Reviews (cont.)

- Early warning about suspicious aspects of the code or design (high complexity measures)

- Opportunities for process improvements
  - Prevention of defects, if lessons are learned in development

- Reduce overall project time scales

- Potential opportunities for communication and learning

Defect Detection Techniques

Examples of Defect Detection:

- Functional
- Structural

Defect Detection Techniques - Functional

Functional:

- Functional testing is based on requirements and functionality.
- Functional testing covers how well the system executes the functions and features.
- Functional testing is not based on any knowledge of internal software design or code.
- Functional testing covers the front-end functions, as well as the back-end operations such as security and how upgrades affect the system.
- Generally functional testing is often done toward the end of the development cycle.
- It is recommended to be started earlier such as individual components and processes.
- Black Box, Acceptance, Closed box and Integration testing are functional testing techniques.

Defect Detection Techniques – Structural

- Structural testing is typically based on the architecture of the system
  - Aspects such as a calling hierarchy, data flow diagram, design specification, etc.
- Structural (white-box) testing may be performed at all test levels
  - System, system integration, or acceptance testing levels (e.g., to business models or menu structures)
- Structural techniques are best used after specification-based techniques

Defect Detection Techniques – Structural

- Structural testing is primarily about coverage
  - Coverage is the degree to which a structure has been exercised by the set of tests
    - Typically it is expressed as a percentage of the items being covered
    - More on coverage later
- Tools can be used at all test levels to measure the code coverage of elements, such as statements or decisions
  - Tools are especially useful at component and component integration testing

Black Box Versus White Box

Black box testing focuses on functional testing, not based on any knowledge of internal software design or code, where as White box focuses on knowledge of the internal logic of an application's code.

Black box testing is based on requirements and functionality, where as White box testing is based on coverage of code statements, branches, paths and conditions.

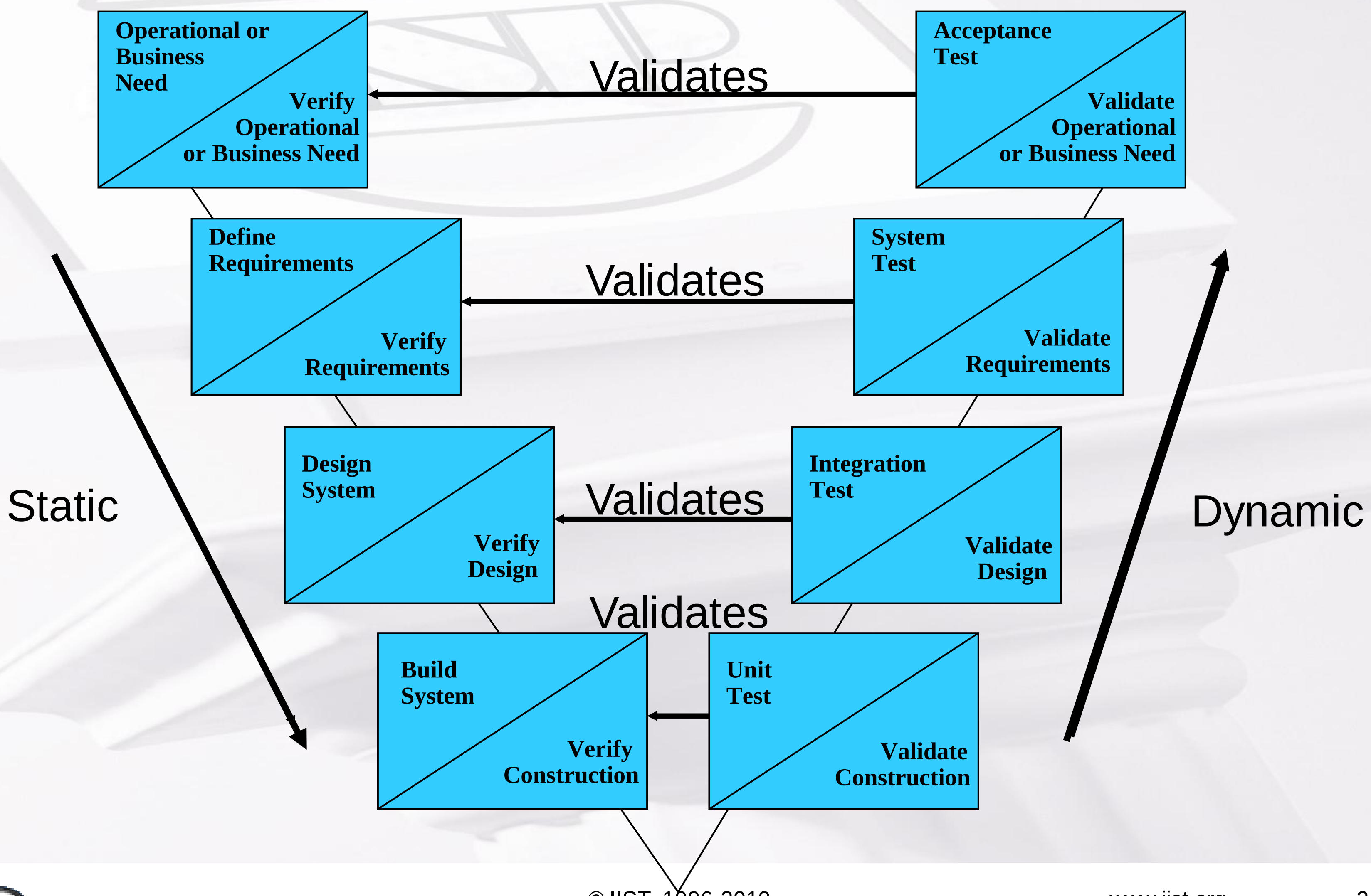
Equivalence Partitioning, Boundary Analysis and Error Guessing are successful techniques for managing the amount of input data for Black box testing.

Statement Coverage, Decision Coverage, Condition Coverage, Decision/Condition Coverage and Multiple Condition Coverage are five important techniques for White box testing.

Static Versus Dynamic

- Static testing is performed using the software documentation. The code is not executing during static testing where as Dynamic testing requires the code to be in an executable state to perform the tests.
- Most verification techniques are static tests where as validation tests are dynamic tests.
- Feasibility Reviews and Requirements Reviews are examples of Static testing.
- Unit, Integrated, System and User Acceptance testing are examples of Dynamic testing.

The "V" Testing Concept



Key Metrics that drive Process Improvement

- Defect Discovery Phase
  - Phase of the SDLC in which the defect was found
- Defect Discovery Method
  - Testing method that found the defect
- Defect Origination Phase
  - Phase of the SDLC in which the defect was originated or “first” introduced into the product
- Root Cause
  - Primary reason why the defect occurred
- Defect Resolution Type
  - How the defect was resolved
- Defect Leakage Reason
  - Root cause to determine method defect escaped the testing phase

Root Cause Examples

Defect Origination Phase

- Concept
- Analyze – (Requirements)
- Design
- Build
- Testing
- Beta
- Deployment
- Production
- Other

Root Cause

- Missing
- Ambiguous/vague
- Wrong
- Changed
- Inconsistent
- Other

Root Cause Examples

Defect Origination Phase

- Concept
- Analyze – (Requirements)
- Design
- Build
- Testing
- Beta
- Deployment
- Production
- Other

Root Cause

- Missing
- Ambiguous/vague
- Logic Wrong
- Logic Changed
- Logic Inconsistent
- Other

Root Cause Examples

Defect Origination Phase

- Concept
- Requirements
- Design
- Build
- Testing
- Beta
- Deployment
- Production
- Other

Root Cause

- Code Error
- Syntax
- Bad data
- Begin-end
- Declaration
- Duplicate code
- Initialization
- Local/global mismatch
- Logic Error
- Memory Usage
- Missing
- Other

Root Cause Examples

Defect Origination Phase

- Concept
- Requirements
- Design
- Build
- Testing
- Beta
- Deployment
- Production
- Other

Root Cause

– Test Data
– Test Environment
– Tester Error
– Test Script Error
– Configuration Mgmt

Effective Execution (Quality Assurance)

Goal: **Effectively execute all of our projects (delivering high-quality software, on budget and on schedule)**

Questions: **Are defects detected early in the SDLC?**

Where are we discovering defects?

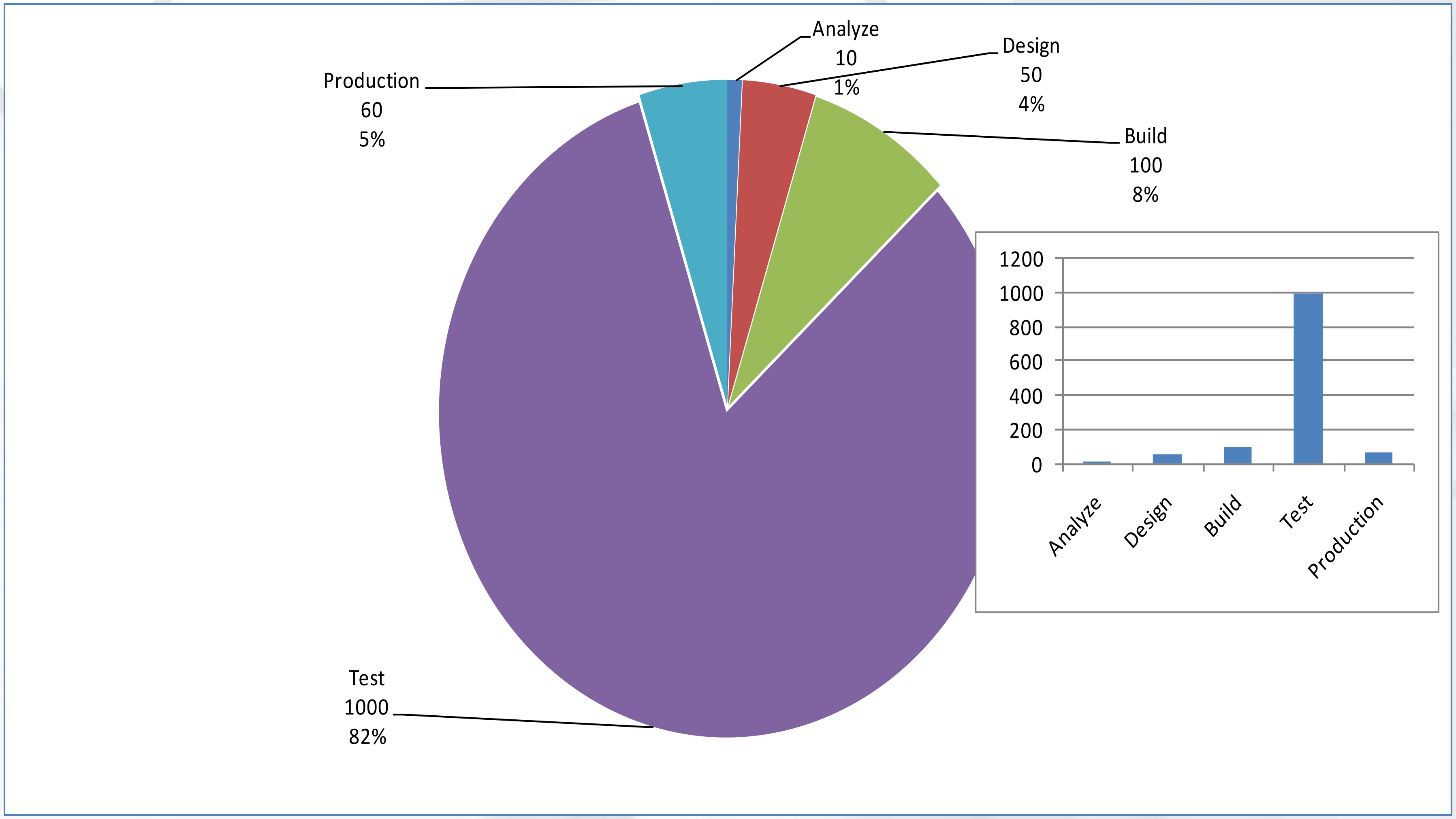
How are we discovering our defects?

Are defects escaping to later test phases?

Metrics:

- Defect detection by discovery phase
- Defect detection by discovery method

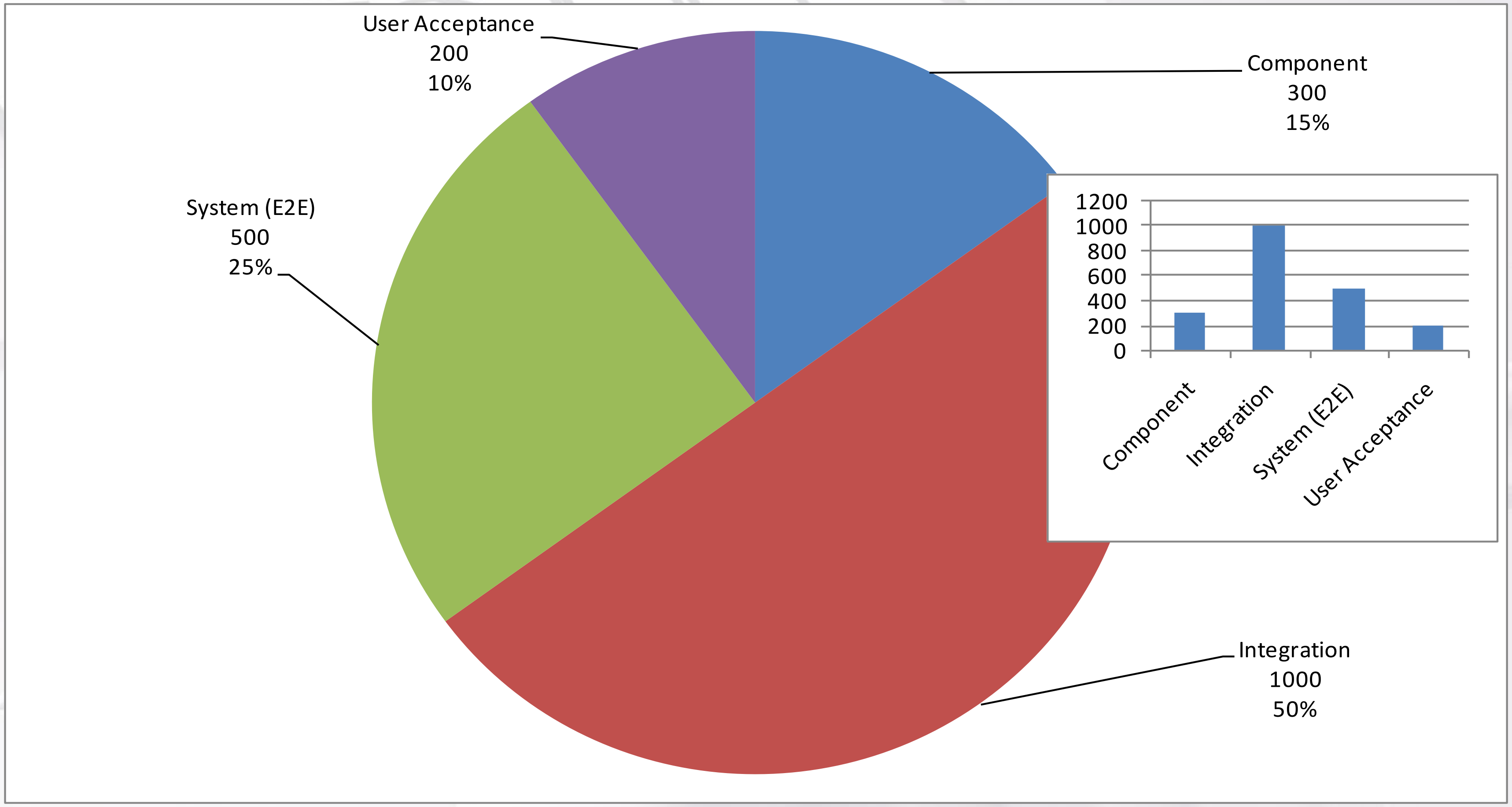
QA Metrics – Defects by Discovery Phase



QA Metrics – Defects by Discovery Phase Key Observations/Recommendations

- 82% of defects are found in Testing
  - Too much pressure on the test team to find defects
  - More defects must be found upstream
  - Recommend more unit testing and/or tracking unit test defects.

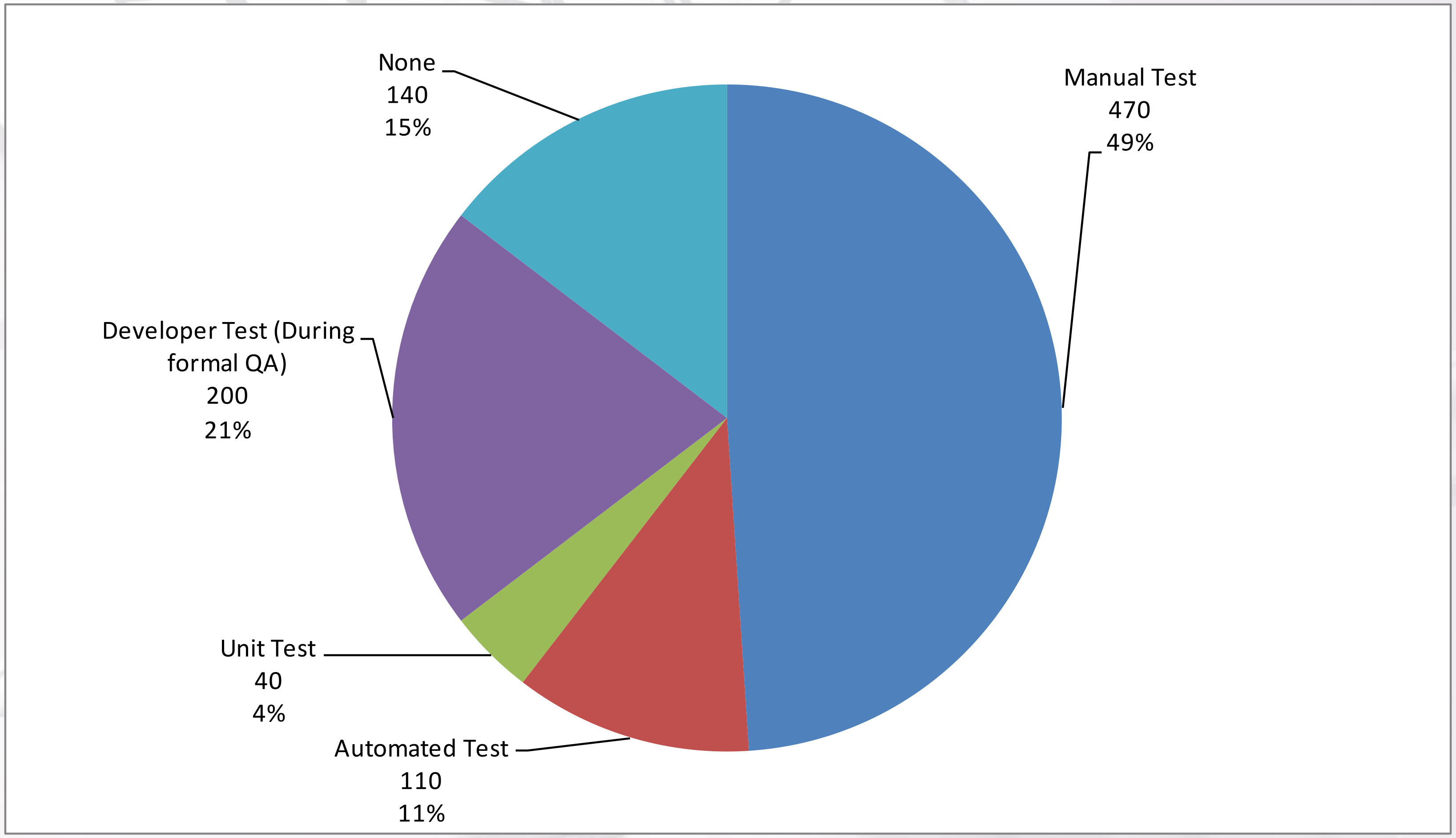
QA Metrics – Defects by Test Phase



QA Metrics – Defects by Test Phase Key Observations/Recommendations

- 50% of defects are found in Integration Testing
  - Is this acceptable?
  - Should more defects be found in Component Test?

QA Metrics – Defects by Discovery Method



QA Metrics – Defects by Discovery Method Key Observations/Recommendations

- Very low % of automated versus manual tests
  - How much are you investing in automated tests?
  - Are you getting your expected ROI on automation?
- How are developers finding such a high % of defects in formal testing?
  - Understand how/what type of testing the developers are doing to find these defects
  - Incorporate these findings into the formal testing process
- 15% (140 Defects) were found but no method was recorded
  - This is valuable data that we are missing
  - Remove the option to select None from your test management tool

Effective Execution (Development)

Goal: **Effectively execute all of our projects (delivering high-quality software, on budget and on schedule)**

Questions: **How effective is our inspection process?**

Where are injecting defects into our products?

How are we resolving defects in our products?

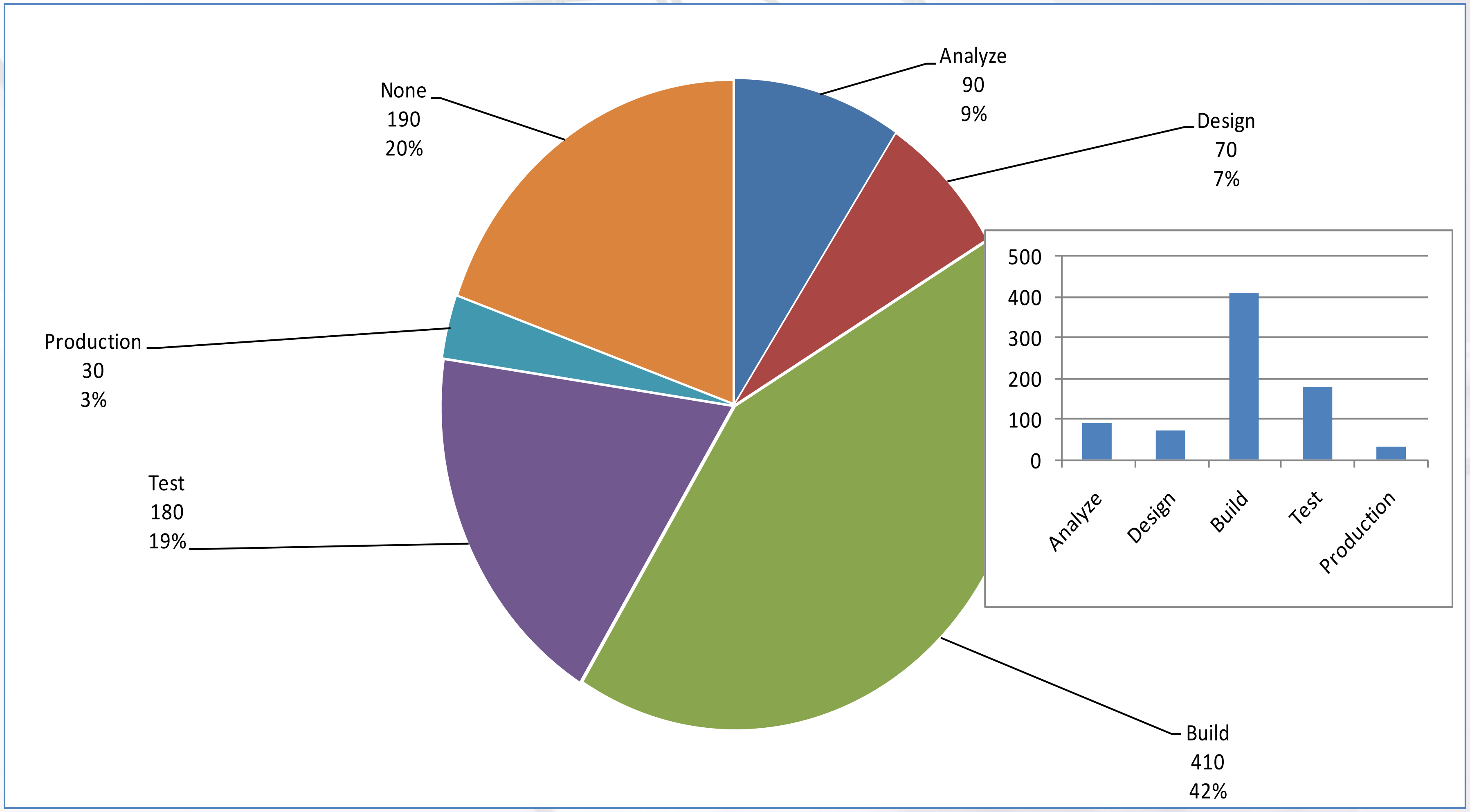
How much time are we spending resolving defects?

What are the primary root causes for our defects?

Metrics:

- Defect detection from Fagan tools
- Defect detection by injection phase
- Defect resolution type
- Total resolution time/Total Verify time
- Defect root cause

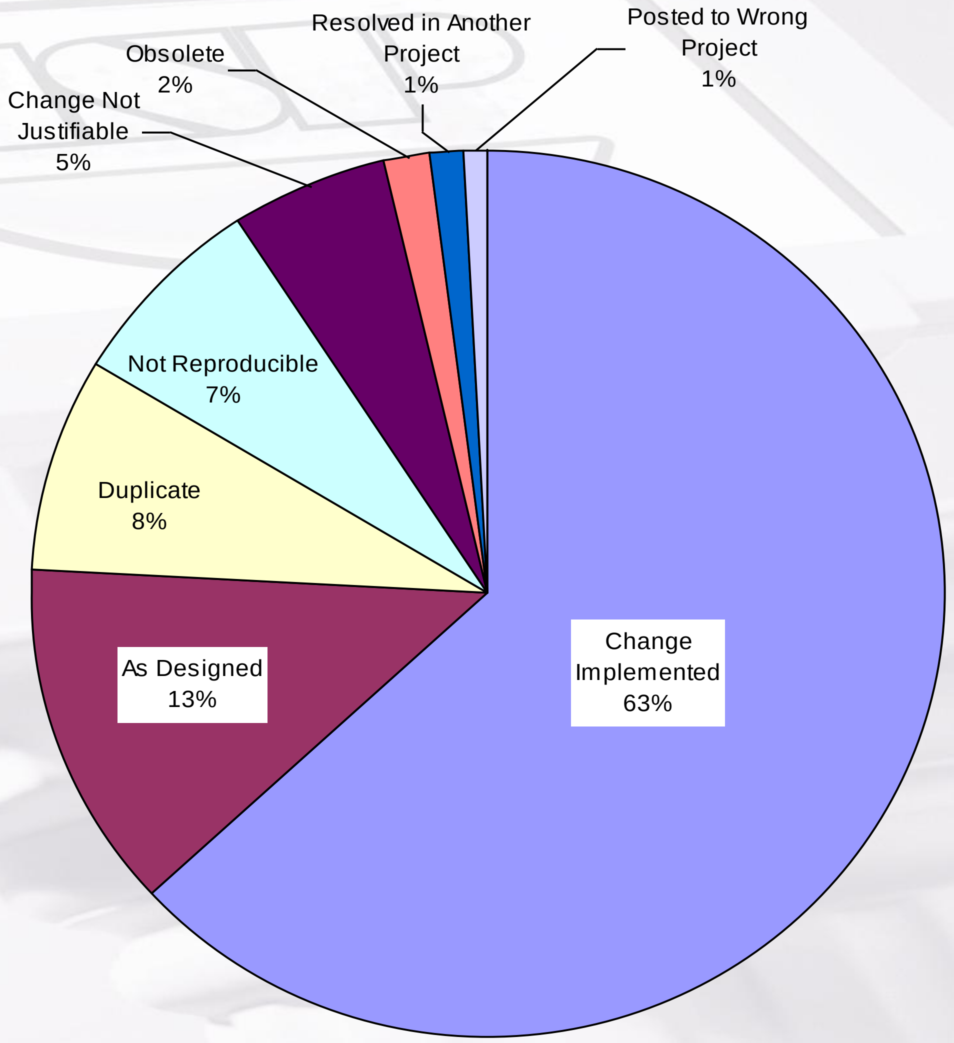
Development Metrics – Defects by Origination Phase



Development Metrics – Defects by Origination Phase
Key Observations/Recommendations

- Note the high % of defects found in Test. Is this accurate?
- 42% of defects originated in Build
  - This is a large number of coding defects
  - Are developers unit testing?
  - Recommend that coding standards, tracking unit test defects and peer reviews
- 9% of defects originated in Analyze? Is this accurate?
- 20% of defects are not accounted for

Development Metrics – Defects by Resolution



Development Metrics – Defects by Resolution Key Observations/Recommendations

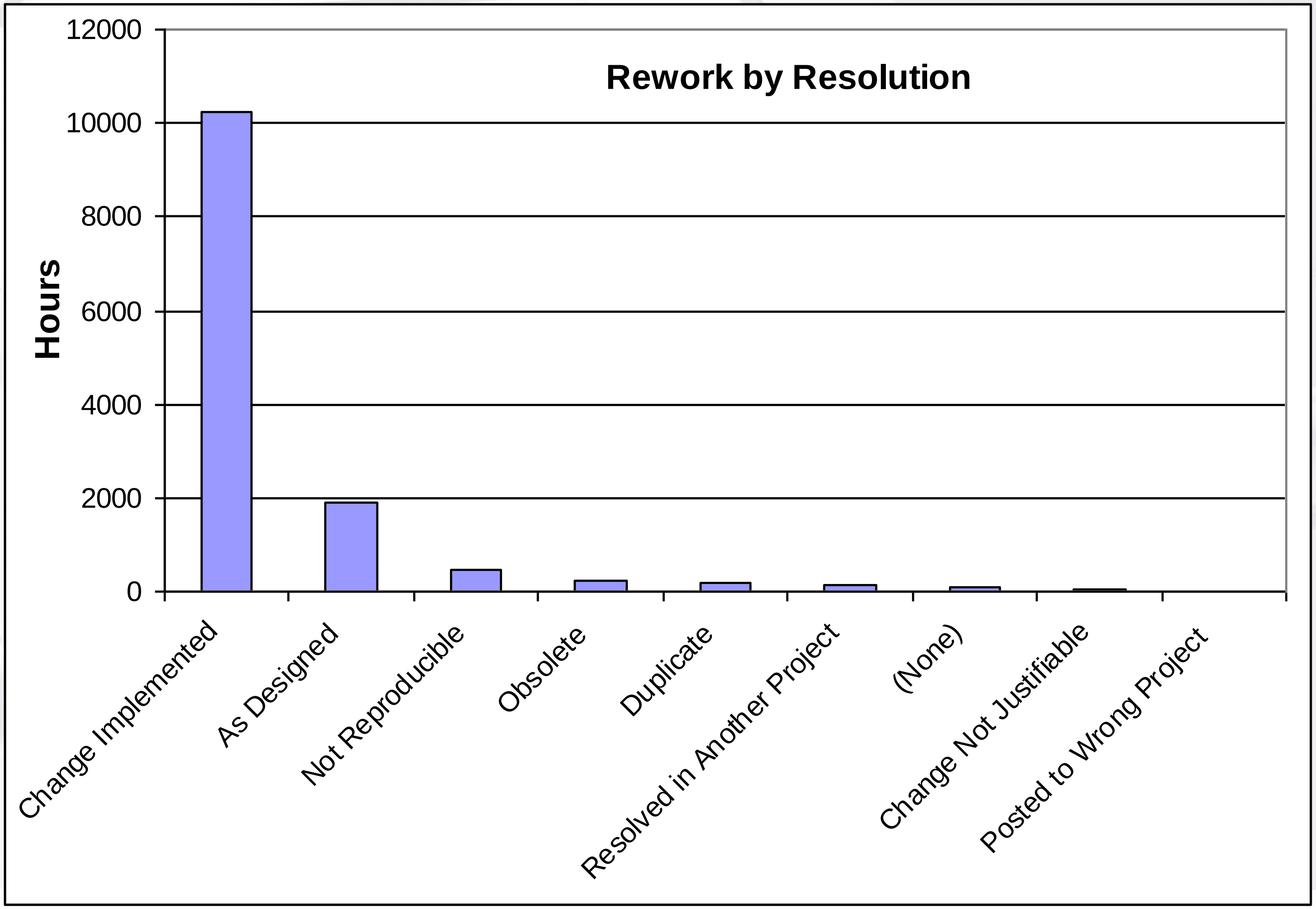
- 13% defects were closed As Designed
  - Is there a disconnect between the specifications and the application? Why are so many defects invalid?
- 8% defects were closed as Duplicate
  - Are testers not communicated with each other or searching for outstanding defects prior to entering their own?
- 7% defects were closed as Not Reproducible
  - Why are so many defects non reproducible? Are there environment or data variables?
  - Is the tester doing something abnormal during the execution of the test?
  - Is the developer trying to reproduce it in a different environment?

Understanding the Cost of Poor Quality (CoPQ)

Cost of Poor Quality:	**The cost of rework associated with fixing any defects in the product. This includes support and maintenance costs associated with a product.**
Cost of Quality:	**The regularly schedule cost for testing a product. This includes unit testing as well as the cost for each test phase.**

REWORK!!

Rework by Resolution

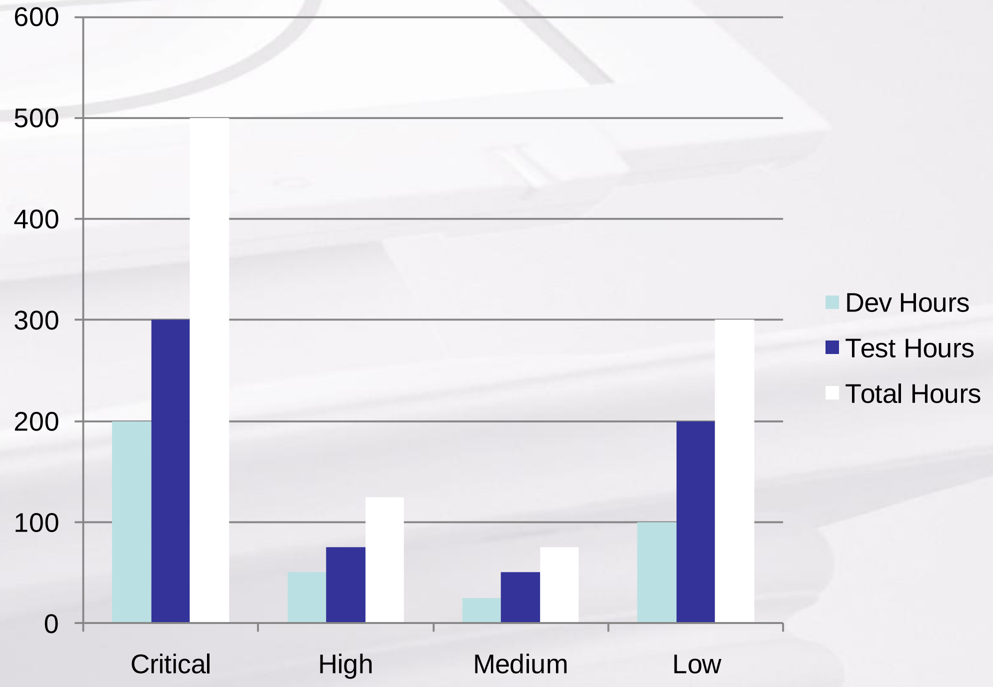


Rework by Resolution
Key Observations/Recommendations

- Over 10K hours were spent fixing defects
  - Is this acceptable?
  - Do you want a further breakdown of this? (see next slide)
- 2K hours were spent “investigating” defects that were invalid due to the application working as designed
  - Is this acceptable?

Rework by Resolution By Severity

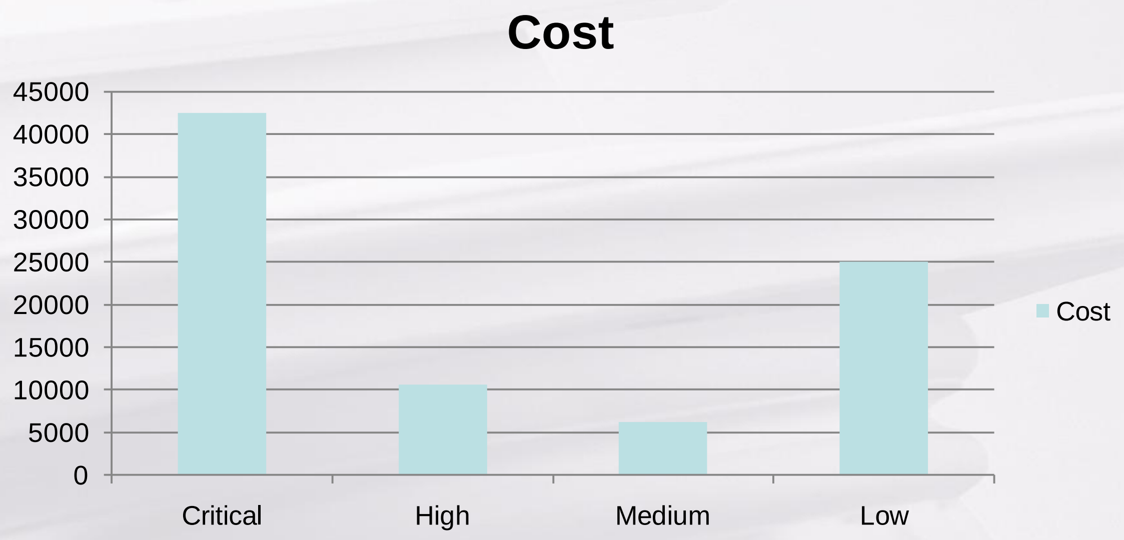
Severity	Dev Hours	Test Hours	Total Hours
Critical	200	300	500
High	50	75	125
Medium	25	50	75
Low	100	200	300



Rework by Resolution By Severity/Cost

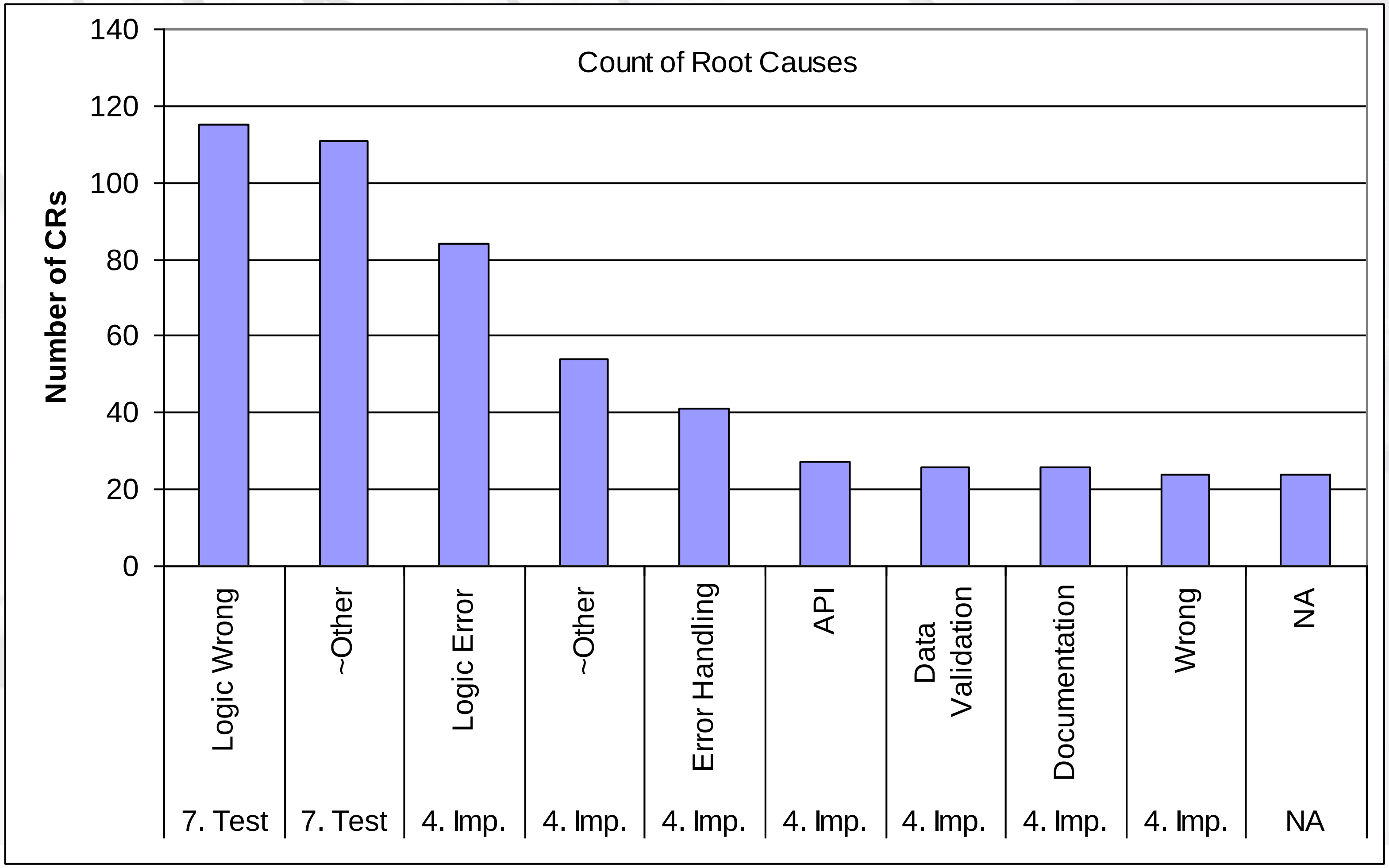
Severity	Dev Hours	Test Hours	Total Hours	Cost
Critical	200	300	500	42500
High	50	75	125	10625
Medium	25	50	75	6250
Low	100	200	300	25000
				84375

Cost:	Dev =	100
	Test =	75



Root Cause Frequency

Fields Used: Origination Phase; Root Cause



Root Cause Frequency
Key Observations/Recommendations

- Almost 200 defects (CRs) are caused due to Logic errors
  - This project is having serious issues with their design logic
  - Are requirements clear?
  - Are these inexperienced developers?
  - Do they truly understand how the system is built?
- Over 170 defects are due to UNKNOWN reasons
  - Unacceptable
  - We have no idea how to fix these issues because the root cause is unknown

THANK YOU!!

QUESTIONS???

- **CONTACT INFO:**

- **Mike.w.ennis@accenture.com**