

# Software Testing

---

## Introduction

# Outline

---

- ❑ Introduction to the subject
  - Quality related activities
  - What is... error/bug/fault/failure/testing?
  - Psychology, goals and testing theory
  - Overview of the testing process – concepts and dimensions
  - Testing principles
  - Standard testing questions
- ❑ Classifying aspects of testing
  - Types
  - Levels

# Approaches...

---

- There are many approaches to software testing:
  - ***Reviews, walkthroughs or inspections*** are considered as **static testing**, whereas actually ***running the program with a given set of test cases*** in a given development stage is referred to as **dynamic testing**.
  - Software testing is used in association with ***verification and validation***

# Validation & Verification

---

- ❑ Software Testing **can not be used to prove** that software works correctly.
- ❑ Software Testing should be viewed as a **risk minimization technique**.
- ❑ Software Testing **still necessary to demonstrate that it conforms to its specification and satisfies** the user's requirements.
  - 2 activities are known as V&V

# Quality related activities

---

- ❑ 3 categories of quality related activities:
  - **Defect Prevention** → prevent poor quality
    - ❑ Early requirements validation and CM
  - **Quality appraisal** → measure quality
    - ❑ Reviews (static testing)
    - ❑ Testing (dynamic testing)
  - **Repair and rework** → correct errors and remove defects.

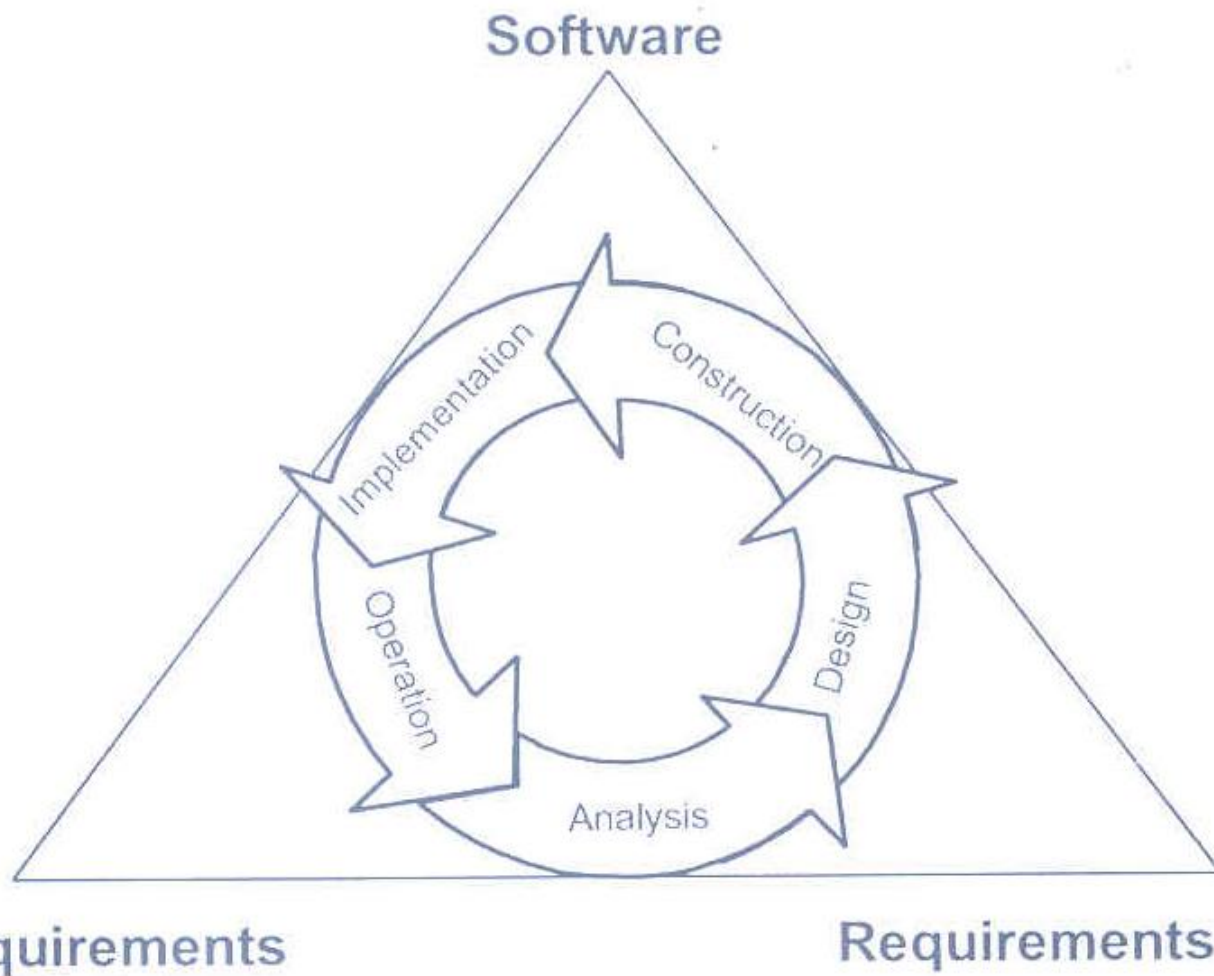


Figure 14: The SDLC and Quality Triangle

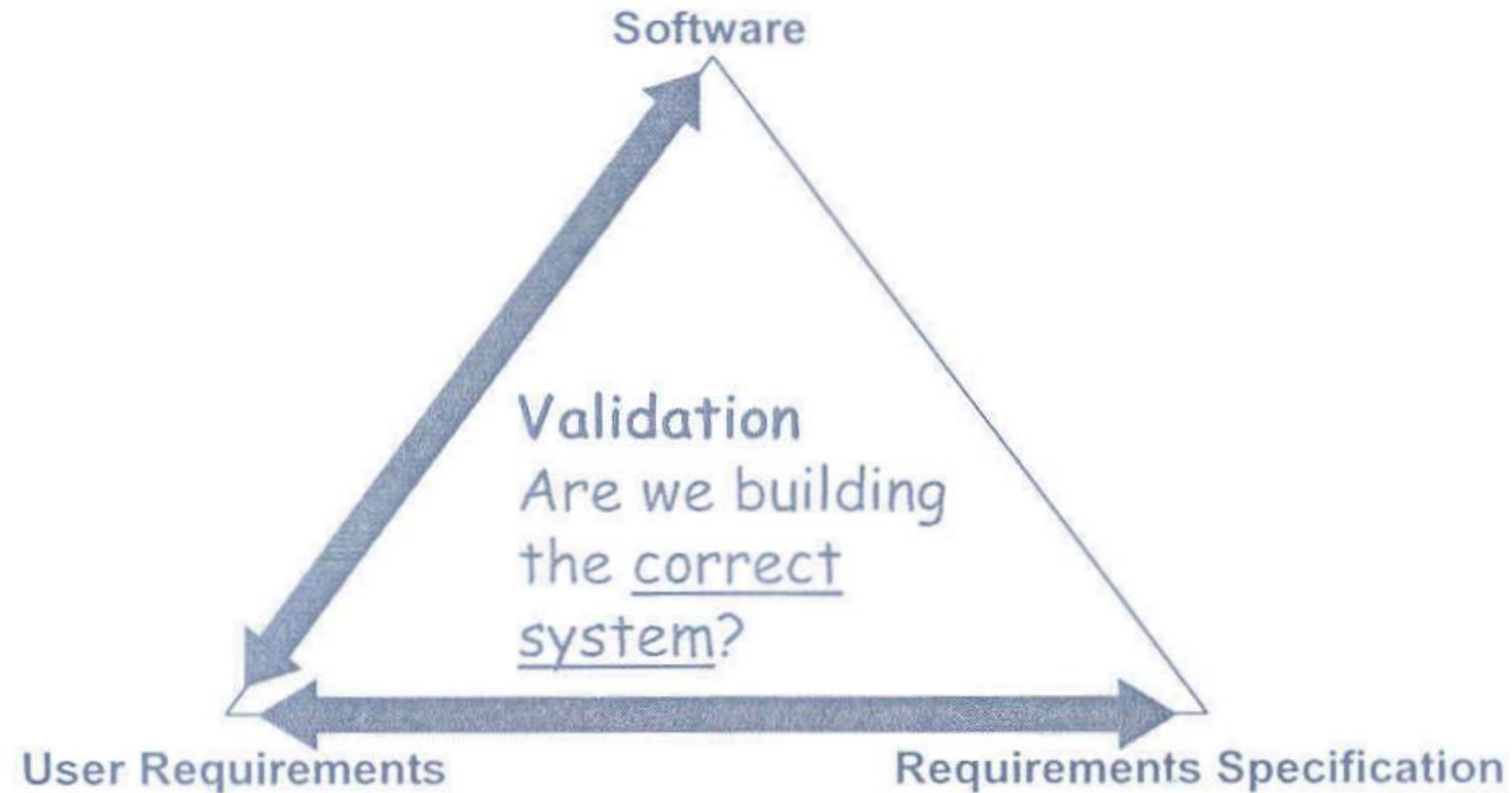


Figure 1: Validation and the Quality Triangle

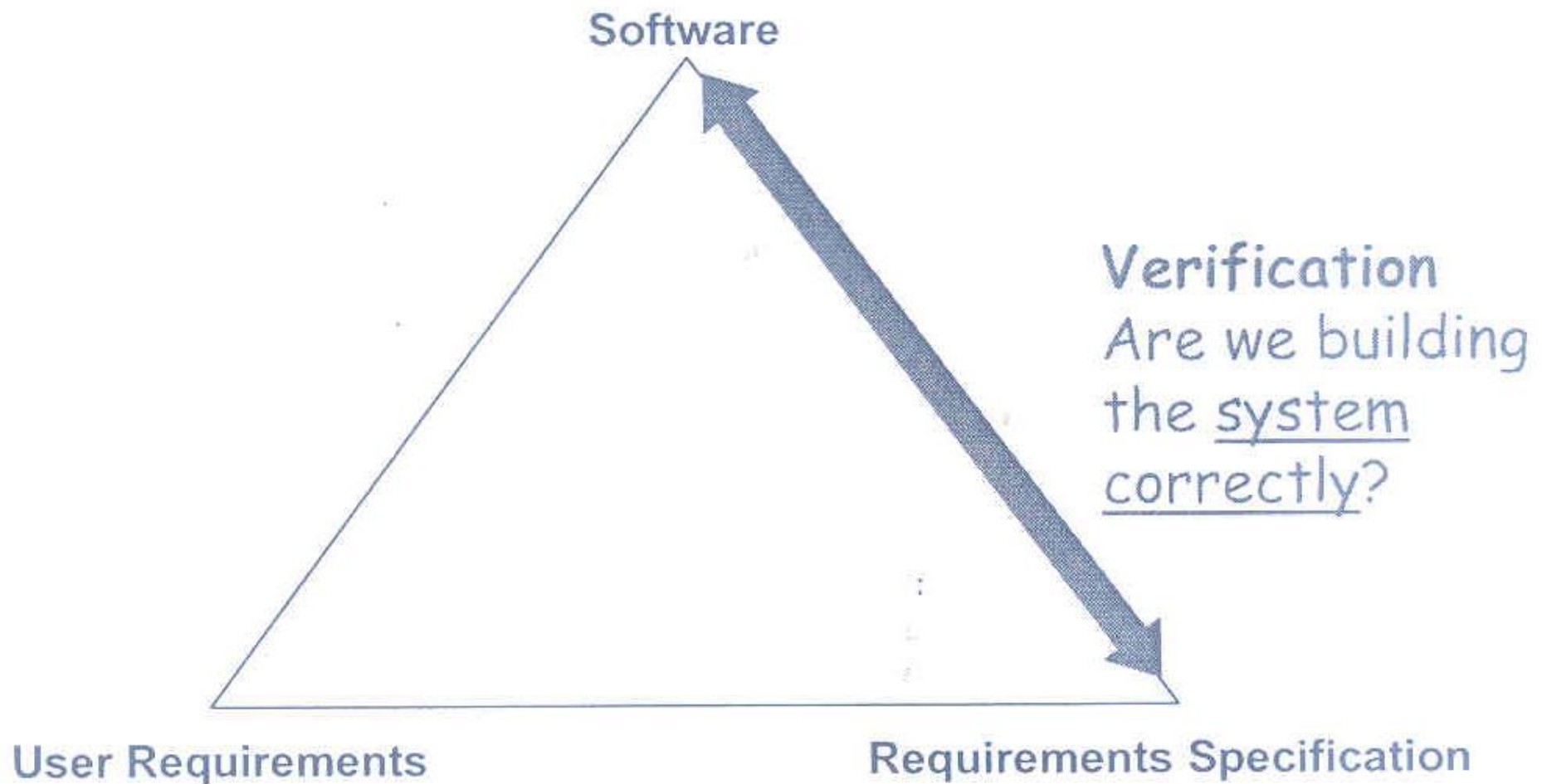


Figure 2: Verification and the Quality Triangle



### 3.3 Early Requirements Validation

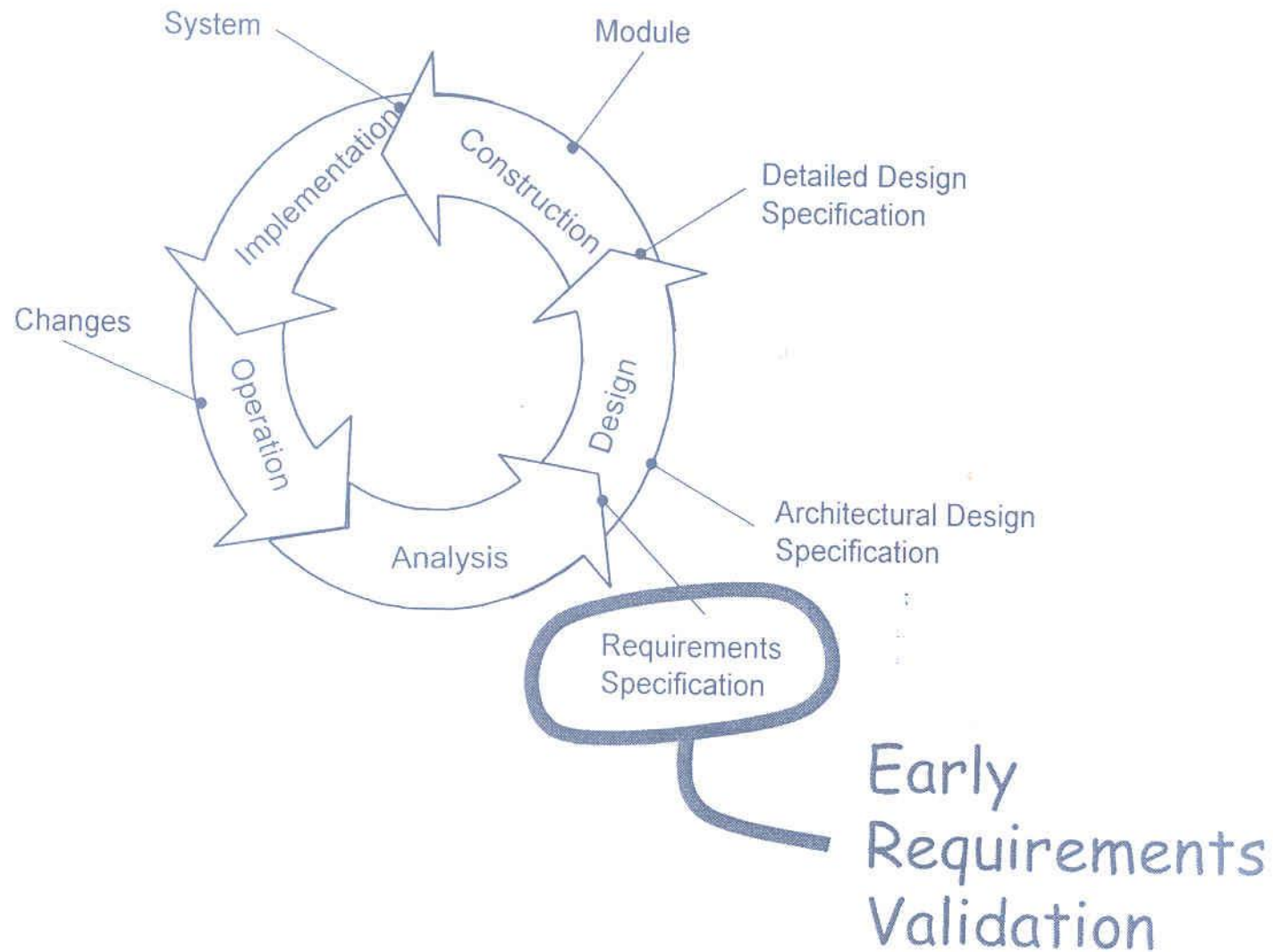


Figure 4: Early Requirements Validation and the SDLC

## 3.4 Configuration Management

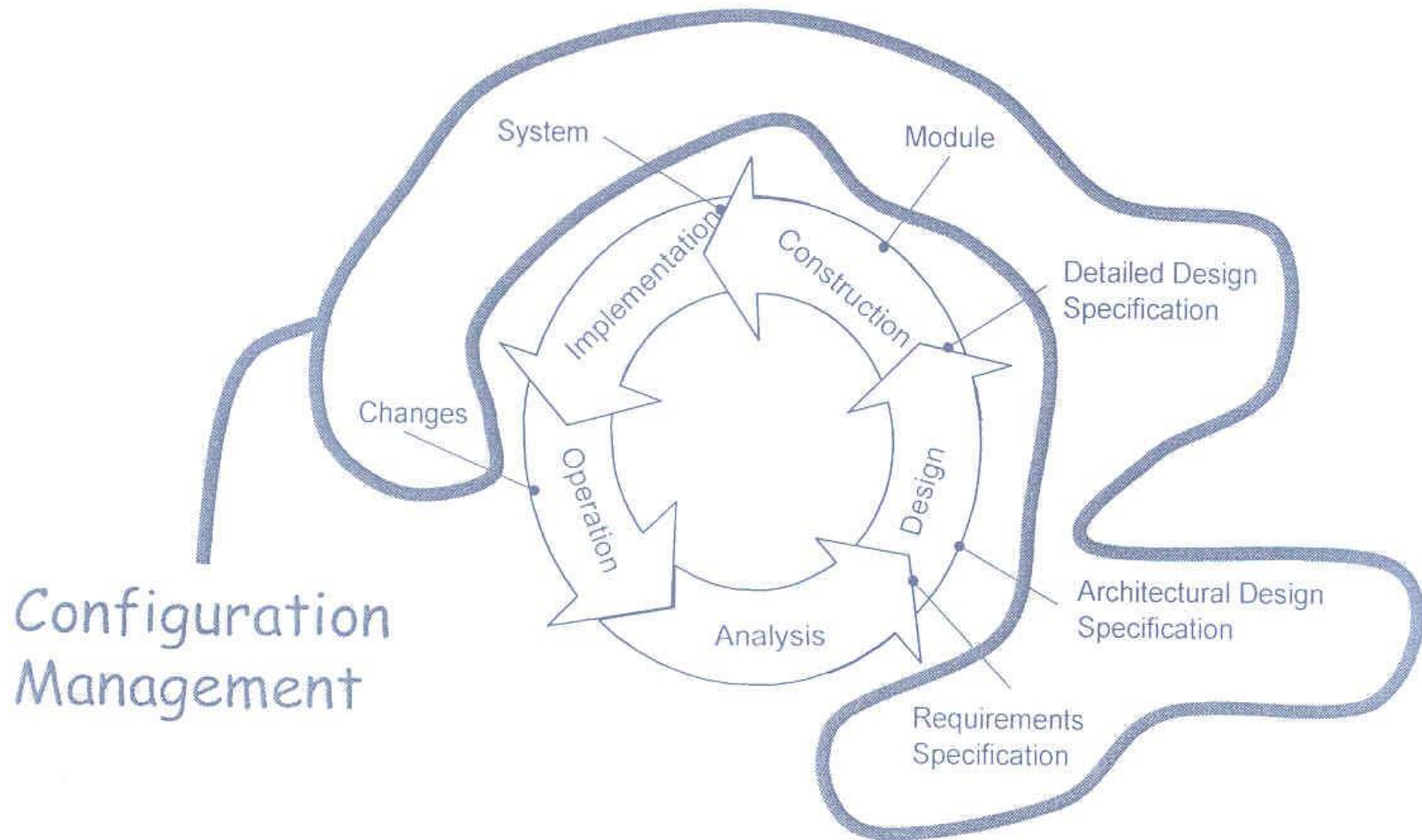


Figure 5: Configuration Management and the SDLC

## 3.5 Reviews

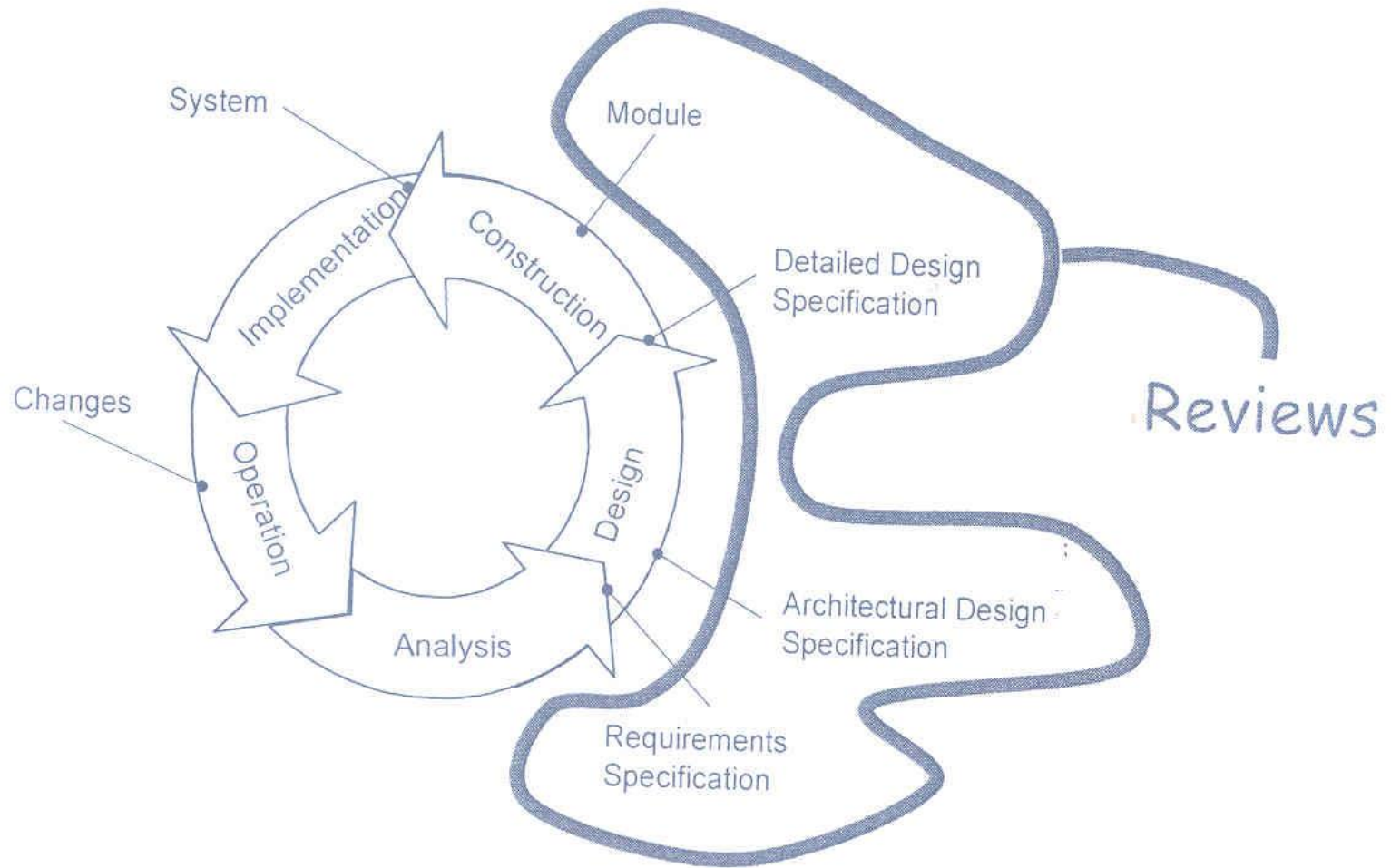


Figure 6: Reviews and the SDLC

## 3.6 Testing

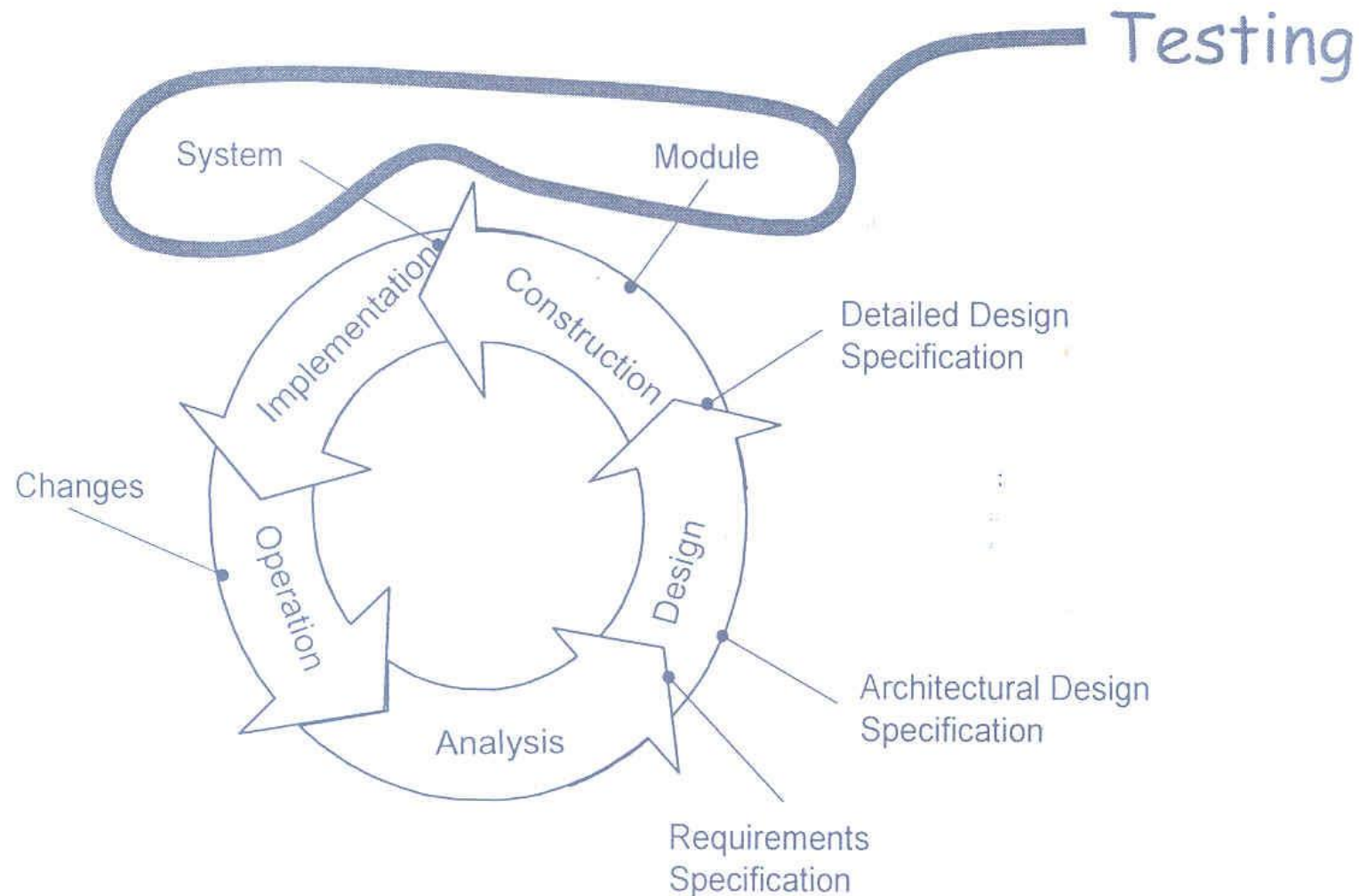


Figure 8: Testing and the SDLC



System

# Test Target

Interface

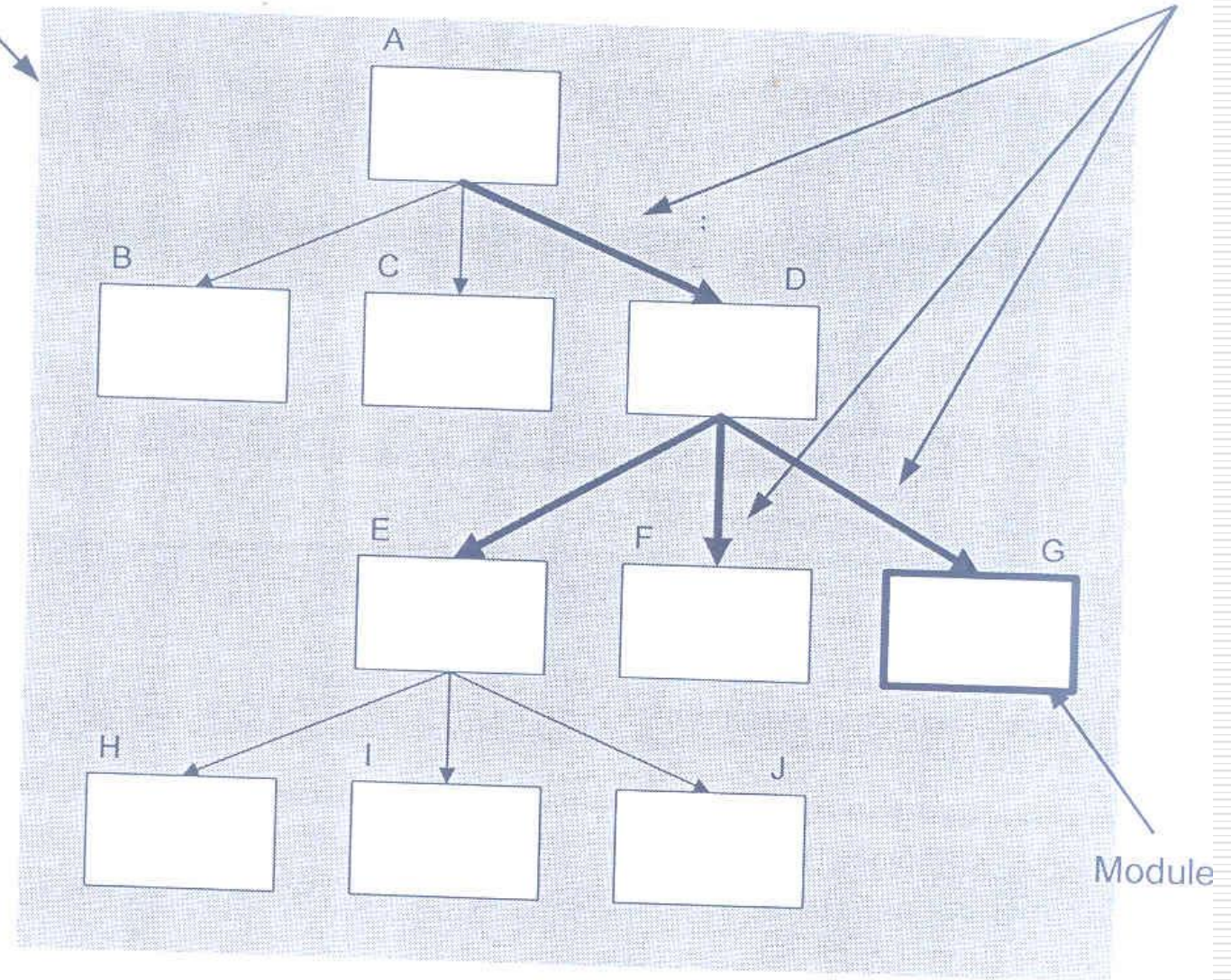


Figure 9: Test Targets

# Validation as a test objective

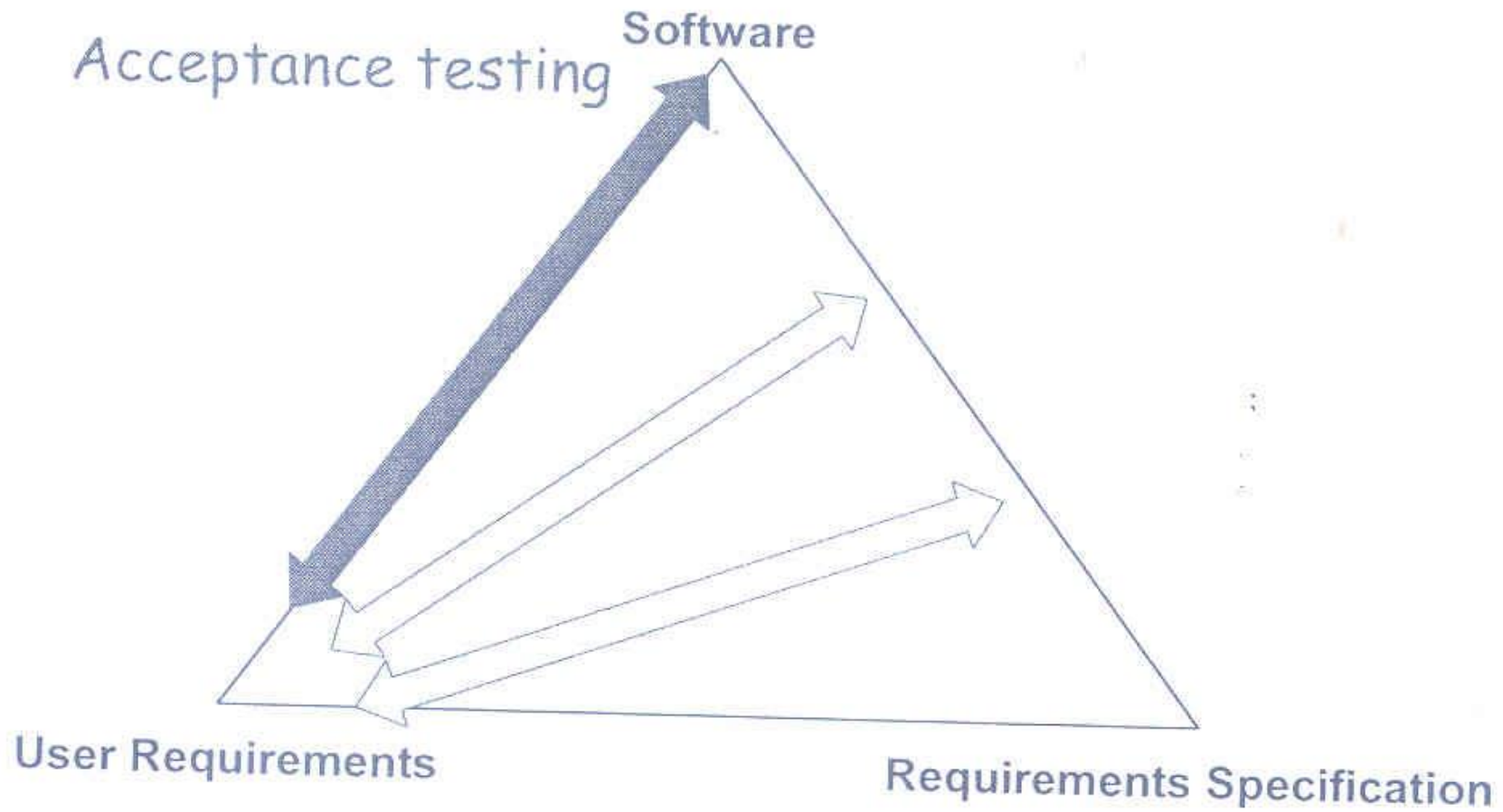


Figure 10: Acceptance Testing

# Verification as a test objective

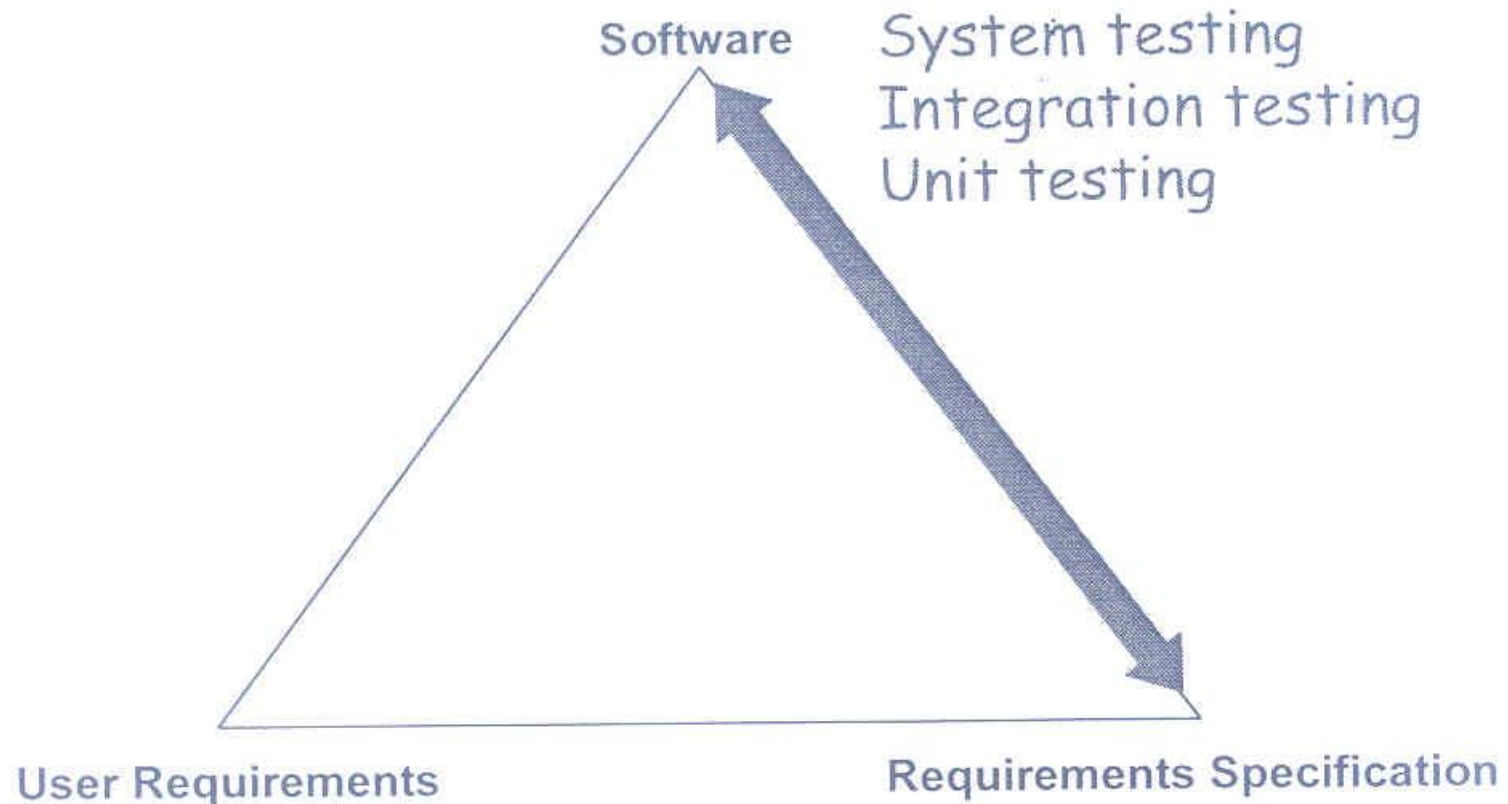


Figure 11: Unit, Integration and Unit Testing



Figure 12: The ISO 9126 Quality Model



# Risk minimization as a test objective

		Software Characteristics																				
		Functionality				Reliability				Usability				Efficiency			Maintainability					
		Suitability	Accuracy	Interoperability	Security	Compliance	Maturity	Fault Tolerance	Recoverability	Compliance	Understandability	Learnability	Operability	Attractiveness	Compliance	Time Behaviour	Resource Behaviour	Compliance	Analysability	Changeability	Stability	Testability
Test	Target																					
Unit Testing	Module																					
Integration Testing	Interface																					
System Testing	System																					
Acceptance Testing																						
Concurrency Testing																						
Background Testing																						
Security Testing																						
Installation Testing																						
Localisation Testing																						
Reliability Testing																						
Recovery Testing																						
Exploratory Testing																						
Usability Testing																						
Performance Testing																						
Stress Testing																						
Regression Testing																						

Figure 12: Risk Minimisation Objectives

# What is... [IEEE]

**error:** something wrong in software itself

**fault:** manifestation of an error

(observable in software behavior)

bug

**failure:** something wrong in software behavior

(deviates from requirements – lệch hướng khỏi y/c)

requirements:

for input  $i$ ,  
give output  $2 \cdot (i^3)$

(so 6 yields 432)

software:

$i = \text{input}(\text{STDIN});$   
 $i = \text{double}(i);$   
 $i = \text{power}(i, 3);$   
 $\text{output}(\text{STDOUT}, i);$

error

output (verbose):

input: 6  
doubling input..  
computing power..  
output: 1728

fault

fault + failure  
18

# The Psychology of Testing

---

- ☐ "Testing is the process of demonstrating that an errors are not present"
- ☐ "Testing is the process of establishing confidence that the program does what it intends to do"
- ☐ "Testing is the process of executing programs with the **intent of finding errors**"
  - Successful (positive) test: exposes an error

# Possible Goals of Testing

---

- Find faults

- Glenford Myers, *The Art of Software Testing*

- Provide confidence

- of reliability
  - of (probable) correctness
  - of detection (therefore absence) of particular faults

# What if testing does not detect any errors

---

- Either
  - the software is high quality
  - the testing process is low quality
- Metrics are required on our testing process if we are to tell which is the right answer.

# What is testing?

---

Executing software/system in a simulated or real environment, using inputs selected *somehow*

# What is testing?

---

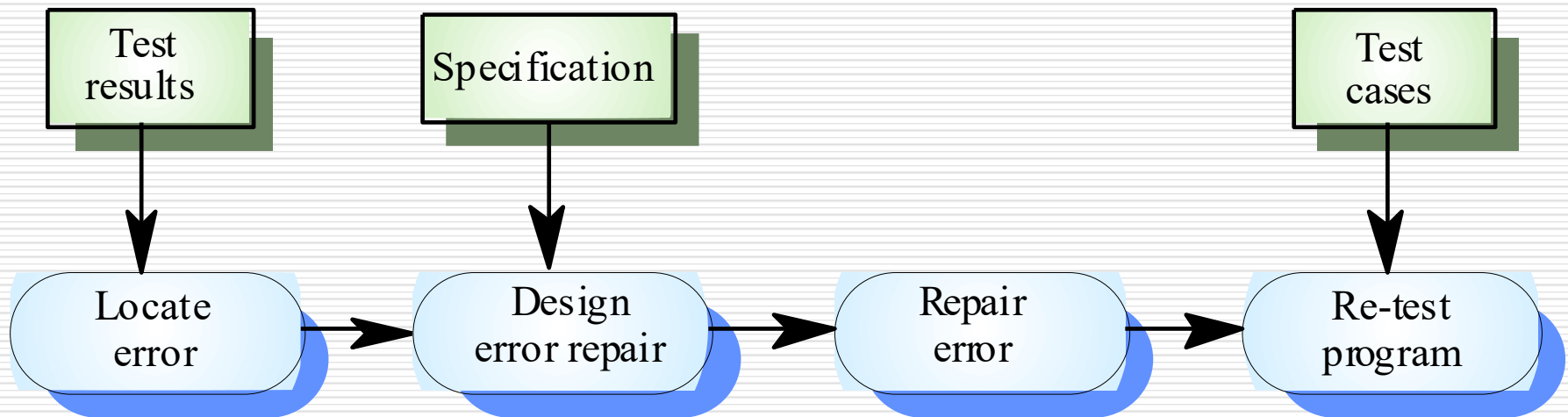
## □ Testing vs. Debugging

- Testing: Finding inputs that cause the software to fail
- Debugging: The process of finding a fault given a failure

# What is testing

---

## □ Debugging process





# Why we must test?

---

- ❑ Testing in the 1980s and 1990s
  - Software engineering was very different
    - ❑ Low quality software was normal and expected
    - ❑ The cost of building better software outweighed the benefits
  - Software was smaller
    - ❑ Most software was bundled, shrink-wrapped, contracted
    - ❑ Industry became dominated by one monopolistic vendor

# Why we must test?

---

## □ Testing in the 21st Century

- The field has dramatically changed
- Today's software market :
  - is much bigger
  - is more competitive
  - has more users
  - used in more places
- The web offers a new deployment platform
  - Very competitive and very available to more users

# Why we must test?

---

## □ Testing in the 21st Century

- Enterprise applications means bigger programs and more users
- Embedded software is now ubiquitous ...
- Paradoxically, free software increases user expectations !

# Why we must test?

---

*Software testing is fast becoming essential to an essential problem in the software industry*

# Why we must test?

---

## □ Future of Software Testing

- Increased specialization in testing teams will lead to more efficient and effective testing
- Testing and QA teams will have more technical expertise
- Developers will have more knowledge about testing and motivation to test better
- Agile processes puts testing first—putting pressure on both testers and developers to test better

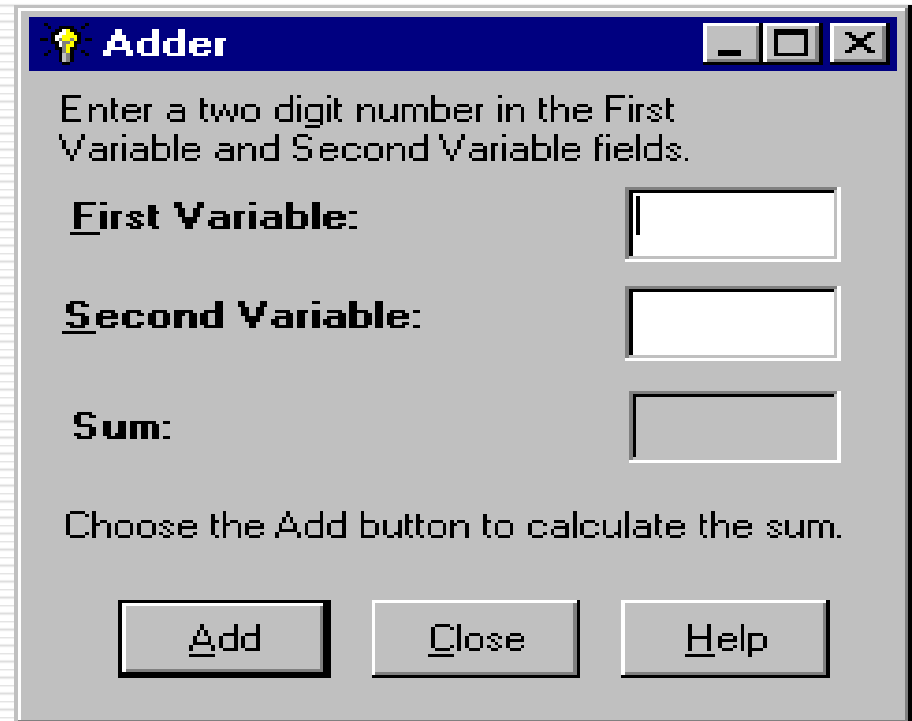
# Why testing is so hard?

---

## □ Example

The program's specification:

- This program is designed to add two numbers, which you will enter
- Each number should be one or two digits



The screenshot shows a classic Windows application window titled "Adder". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains the following elements:

- A text instruction: "Enter a two digit number in the First Variable and Second Variable fields."
- Two input fields: The first is labeled "First Variable:" and the second is labeled "Second Variable:". Both are empty text boxes.
- A third input field labeled "Sum:" is positioned below the others.
- A text instruction: "Choose the Add button to calculate the sum."
- Three buttons at the bottom: "Add", "Close", and "Help". The "Add" button is highlighted with a black border.

# Why testing is so hard?

---

- ❑ Determining whether or not outputs are correct.
- ❑ Comparing resulting internal states to expected states.
- ❑ Determining whether adequate testing has been done.
- ❑ Determining what you can say about the software when testing is completed.
- ❑ Measuring performance characteristics.

# Why testing is so hard?

---

- ❑ Testing resources (staff, time, tools, labs) are limited.
- ❑ Specifications are frequently unclear/ambiguous and changing (and not necessarily complete and up-to-date).
- ❑ Systems are almost always too large to permit test cases to be selected based on code characteristics.



# QA Vs QC

---

☐ Your ideas ???

# Three themes prevail:

---

- ❑ Quality Assessment (What percentage of defects are captured by our testing process, how many remain?)
- ❑ Risk Management (What is the risk related to remaining defects?)
- ❑ Test Process Improvement (How long does our testing process take?)

# What is...

---

## testing:

by experiment,

- ■ find errors in software (Myers, 1979)
- ■ establish quality of software (Hetzel, 1988)

## a succesful test:

- ■ finds at least one error
- ■ gives quality judgment (phán quyết) with maximum confidence with minimum effort

# Testing Theory (such as it is)

---

- Plenty of negative results
  - Nothing *guarantees* correctness
  - Statistical confidence is prohibitively expensive
  - Being systematic may not improve fault detection
    - as compared to simple random testing
  - So what did you expect, decision procedures for undecidable problems?

# What's been said?

---

- Dijkstra:

  - Testing can show the presence of bugs, but not the absence

- Beizer:

  - 1<sup>st</sup> law: Every method you use to prevent or find bugs leaves a residue of subtler bugs (xót lại những lỗi không dễ phát hiện)

  - 2<sup>nd</sup> law: Software complexity grows to the limits of our ability to manage it

- Beizer:

  - Testers are not better at test design than programmers are at code design

- ...

- Let's start with a global look at the testing process

# Question

---

- ☐ What are the objectives of testing ?
- ☐ What is a success testing ?

# Objectives of Testing

---

**The Objective of Testing a Program is to Find Problems.**

**Finding problems is the core of your work. You should want to find as many problems as possible. The more serious the problem tester find, the better tester is.**

***A test that reveals a problem is a success. A test that did not reveal a problem is (often) a waste of time!***

# Objectives of Testing

---

The Purpose of Finding Problems  
is to Get Them Fixed.

The point of the exercise is quality  
improvement!

- ❑ ***The best tester is not the one who finds the most bugs or who embarrasses the most programmers.***
- ❑ ***The best tester is the one who gets the most bugs fixed.***



# Question

---

- ☐ List some tasks of testing team ?

# Testing group

---

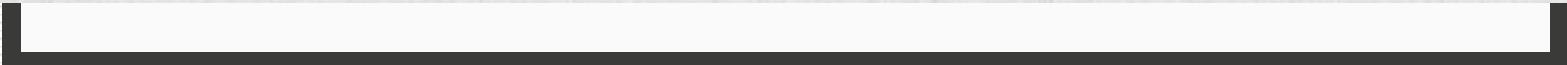
- ❑ The Testing Group provides important technical services, including:
  - Finding and reporting bugs.
  - Identifying weak areas of the program.
  - Identifying high risk areas of a project.
  - Explaining findings in ways that help
    - ❑ Engineering solve or fix the problems.
    - ❑ The customer service staff help customers
    - ❑ Management make sound business decisions about each bug.

# Testing group

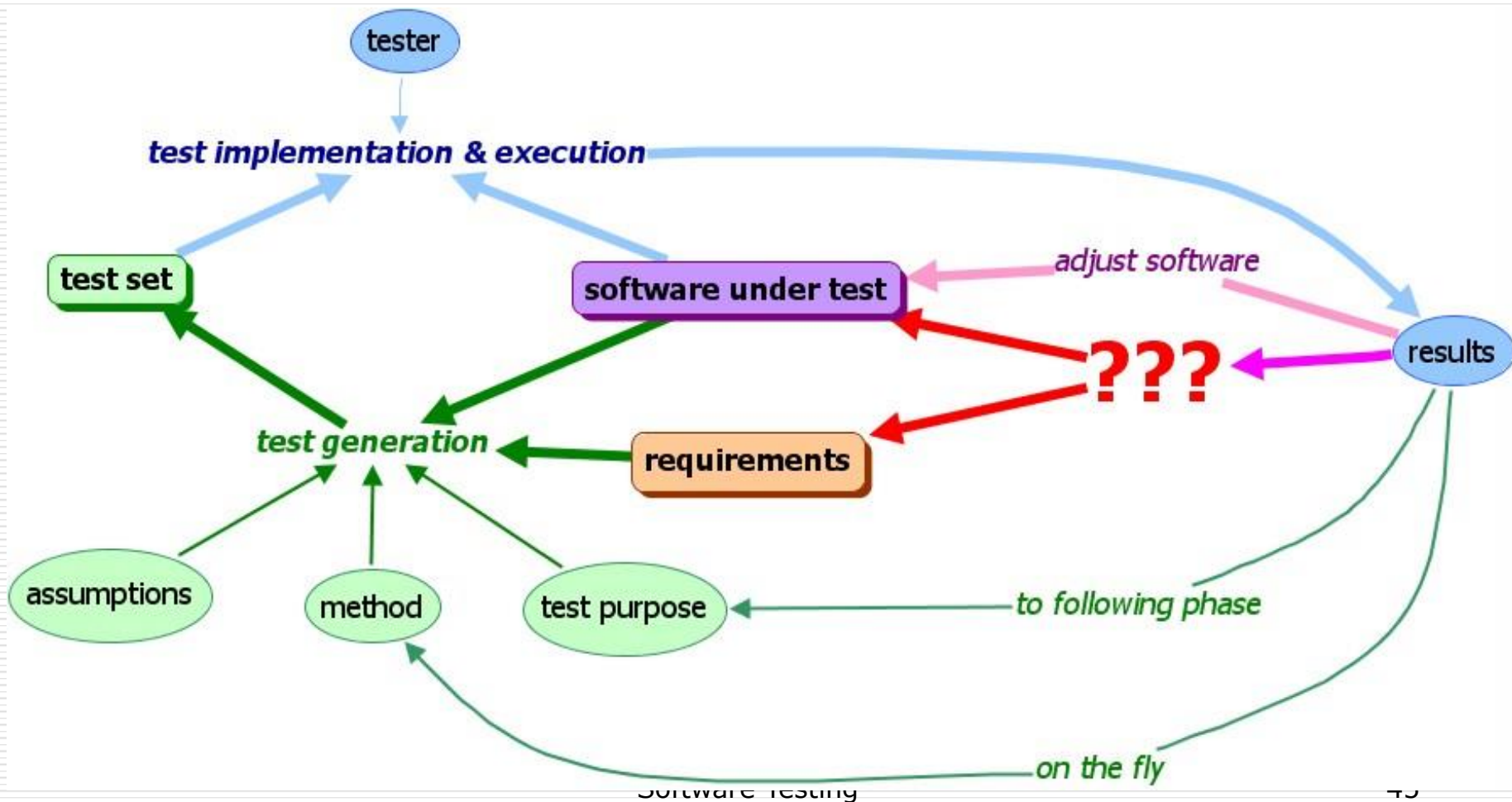
---

- Find Problems
  - Find bugs.
  - Find design issues.
  - Find more efficient ways to find bugs.
  
- Communicate Problems
  - Report bugs and design issues.
  - Report on testing progress.
  - Evaluate and report the program's stability.
  
- More Senior Testers Manage/Supervise Testing Projects
  - Prepare test plans and schedules.
  - Estimate testing tasks, resources, time and budget.
  - Measure and report testing progress against milestones.
  - Teach other testers to find bugs.

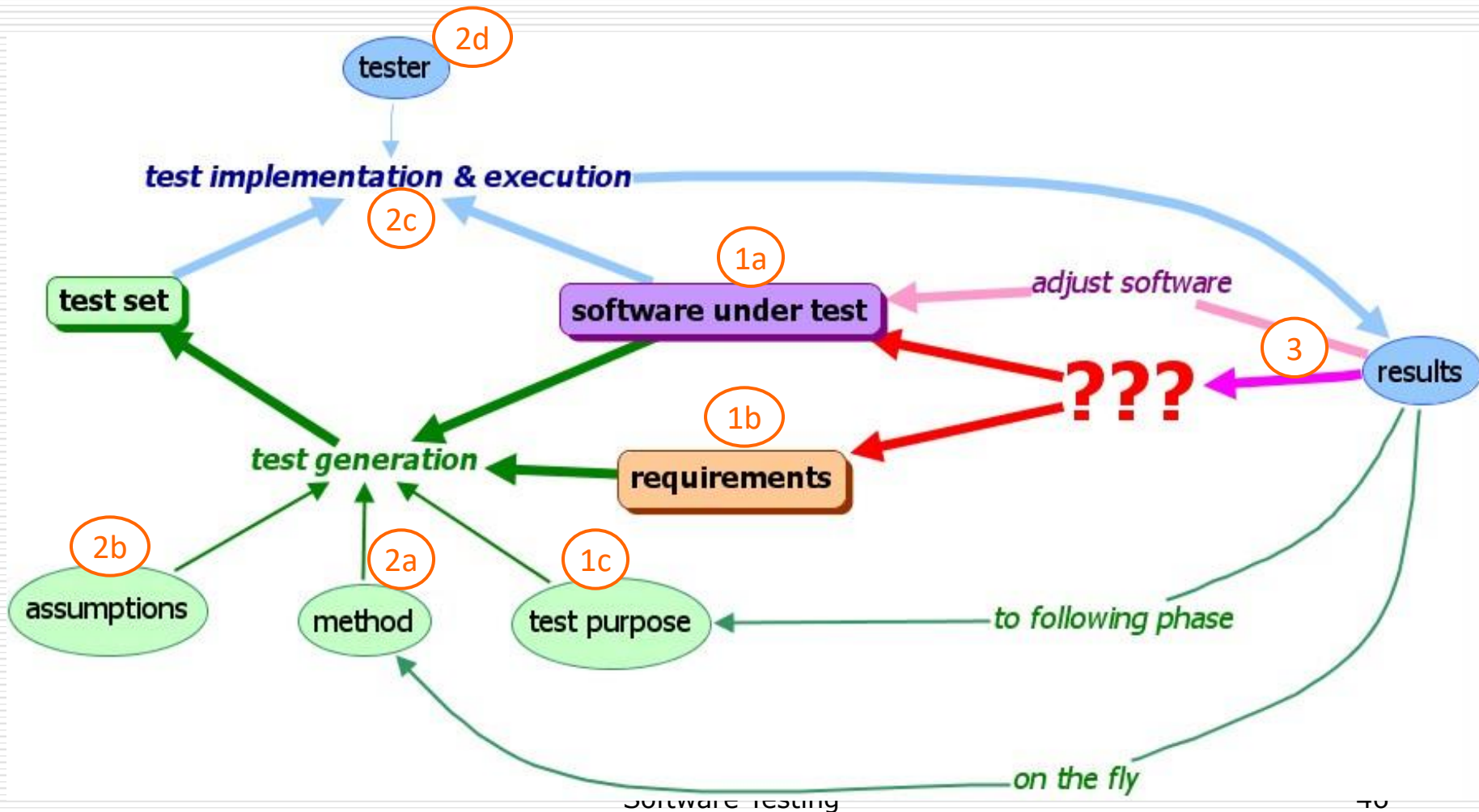
<p> <b>1</b> <b>2</b> <b>3</b> <b>4</b> <b>5</b> <b>6</b> <b>7</b> <b>8</b> <b>9</b> <b>10</b> <b>11</b> <b>12</b> <b>13</b> <b>14</b> <b>15</b> <b>16</b> <b>17</b> <b>18</b> <b>19</b> <b>20</b> <b>21</b> <b>22</b> <b>23</b> <b>24</b> <b>25</b> <b>26</b> <b>27</b> <b>28</b> <b>29</b> <b>30</b> <b>31</b> <b>32</b> <b>33</b> <b>34</b> <b>35</b> <b>36</b> <b>37</b> <b>38</b> <b>39</b> <b>40</b> <b>41</b> <b>42</b> <b>43</b> <b>44</b> <b>45</b> <b>46</b> <b>47</b> <b>48</b> <b>49</b> <b>50</b> <b>51</b> <b>52</b> </p>	<p> <b>53</b> <b>54</b> <b>55</b> <b>56</b> <b>57</b> <b>58</b> <b>59</b> <b>60</b> <b>61</b> <b>62</b> <b>63</b> <b>64</b> <b>65</b> <b>66</b> <b>67</b> <b>68</b> <b>69</b> <b>70</b> <b>71</b> <b>72</b> <b>73</b> <b>74</b> <b>75</b> <b>76</b> <b>77</b> <b>78</b> <b>79</b> <b>80</b> <b>81</b> <b>82</b> <b>83</b> <b>84</b> <b>85</b> <b>86</b> <b>87</b> <b>88</b> <b>89</b> <b>90</b> <b>91</b> <b>92</b> <b>93</b> <b>94</b> <b>95</b> <b>96</b> <b>97</b> <b>98</b> <b>99</b> <b>100</b> </p>
---	---



# Concept map of the testing process



# Dimensions + concept map



# Dimensions of software testing

---

## 1. What is tested?

- a. Software characteristics (design/embedded?/language/...)
- b. Requirements (what property/quality do we want from the software)
- c. Purpose (what development phase, what kind of errors, what next step)

## 2. How to test?

- a. Method (adequacy criterion: how to generate how many tests)
- b. Assumptions (limitations, simplifications, heuristics)
- c. Organization (documentation, implementation, execution:platform, interactive, batch)
- d. Who tests? (programmer, user, third party)

## 3. How are the results evaluated?

# What information can we exploit?

## 1b. Requirements

---

### ☐ functional:

- the behavior of the system is correct
- precise

### ☐ non-functional:

- performance, reliability, compatibility, robustness (stress/volume/recovery), usability, ...
- possibly quantifiable, always vague



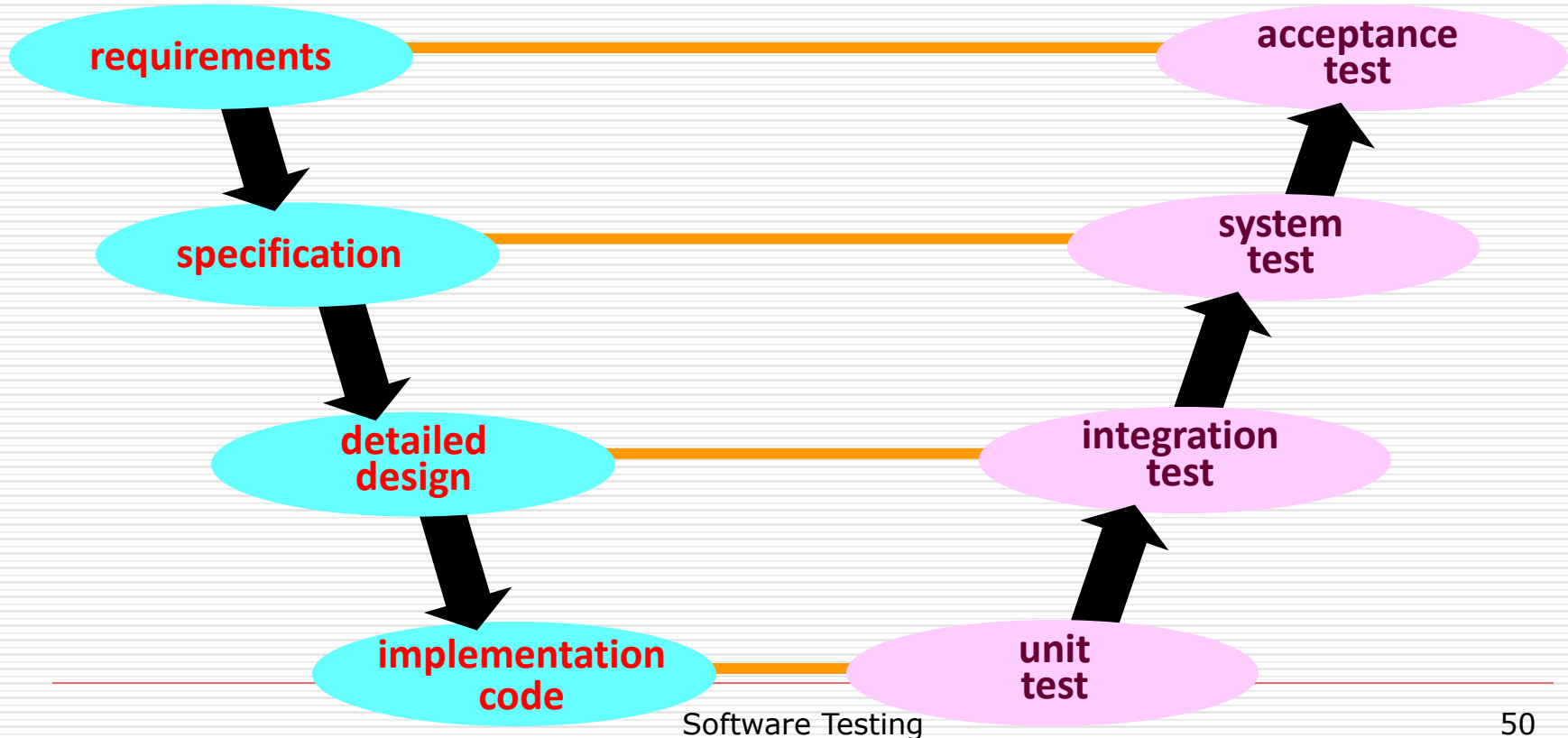
# What other information can we exploit?

---

- ❑ Specifications (formal or informal)
  - in Oracles
  - for Selection, Generation, Adequacy
- ❑ Designs
  - ...
- ❑ Code
  - for Selection, Generation, Adequacy
- ❑ Usage (historical or models)
- ❑ Organization experience

# 1c. Test purpose

Software development phases (V-model):



# 1c. Test purpose

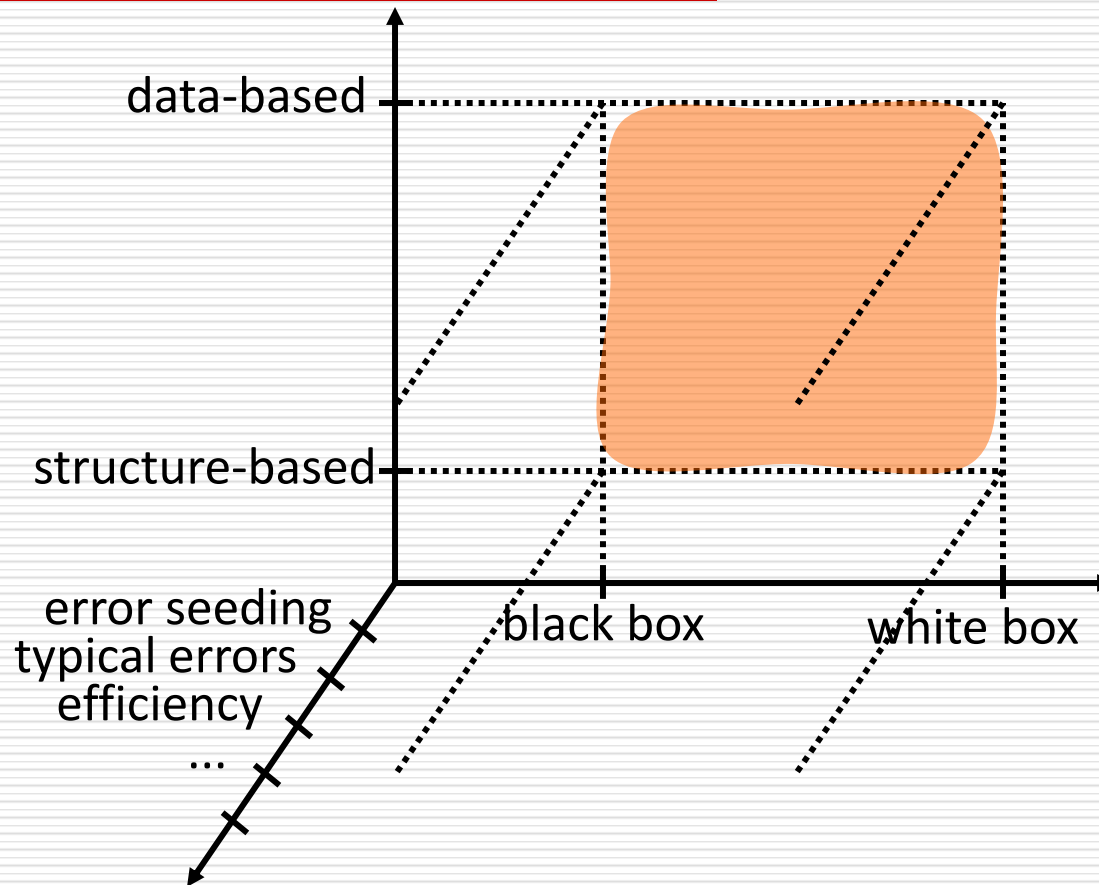
---

What errors to find?

- ☐ typical typos
- ☐ functional mistakes
- ☐ unimplemented required features
  - 'software does not do all it should do'
- ☐ implemented non-required features
  - 'software does things it should not do'

## 2a. Test method dimensions

---



## 2b. Assumptions, limitations

---

- ☐ single/multiple fault:
  - clustering/dependency of errors
- ☐ testability:
  - can software be tested?
    - time, resources, accessibility,
- ☐ perfect repair
- ☐ ...

## 2c. Test organization

---

- ❑ Documentation
  - for reuse!
- ❑ Implementation
  - platform
  - batch?
  - inputs, coordination, ...
- ❑ Execution
  - duration, timing, interruptions
  - manual/interactive or automated
  - in parallel on several systems

# Testing Principles

---

- ❑ Test case must include the definition of expected results
- ❑ A program should not test her/his own code
- ❑ A programming organization should not test its own programs
- ❑ Thoroughly inspect the results of each test
- ❑ Test cases must be also written for invalid inputs

# Testing Principles

---

- ❑ Check that programs do not do unexpected things
- ❑ Test cases should not be thrown away
- ❑ Do not plan testing assuming that there are no errors
- ❑ The probability of error in a piece of code is proportional (tương ứng) to the errors found so far in this part of the code



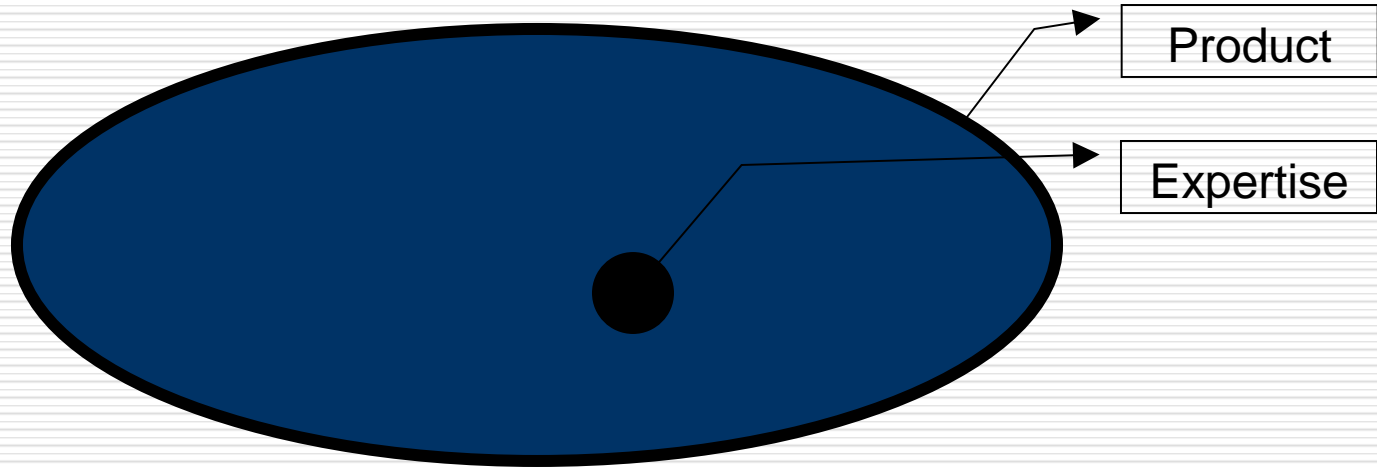
# Test Groups – Pros

---

- ❑ Only people to use a system heavily is experts;
- ❑ Independent testing is typically more efficient at detecting defects related to special cases, interaction between modules, and system level usability and performance problems
- ❑ Programmers are neither trained, nor motivated to test
- ❑ Overall, more of the defects in the product will likely be detected.
- ❑ Test groups can provide insight into the reliability of the software before it is actually shipped

# Developer's Mindset

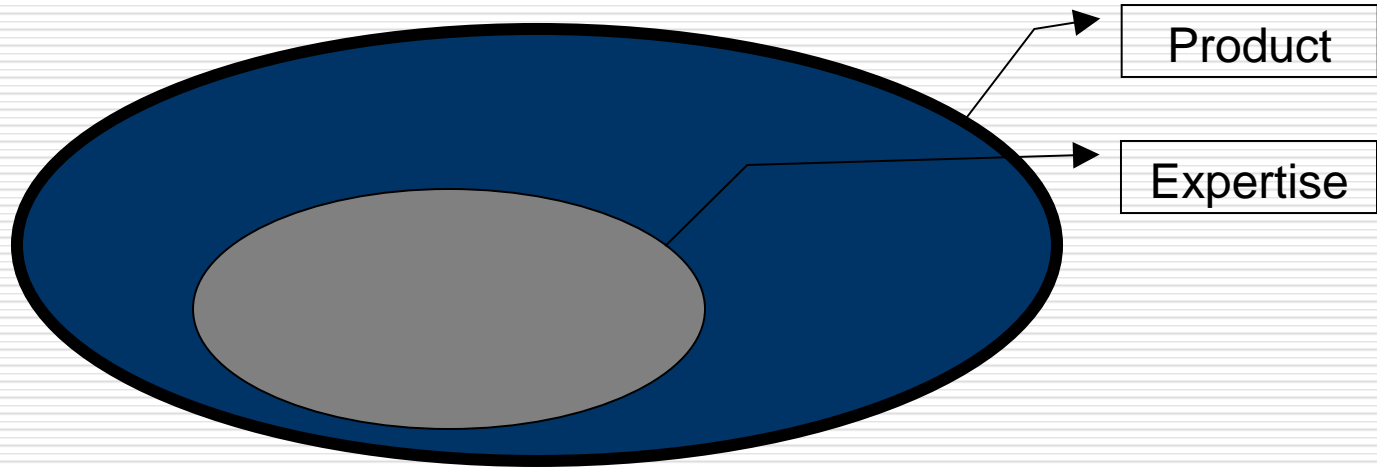
---



- ❑ A good developer is clever
- ❑ Emphasis on depth of expertise

# Tester's Mindset

---



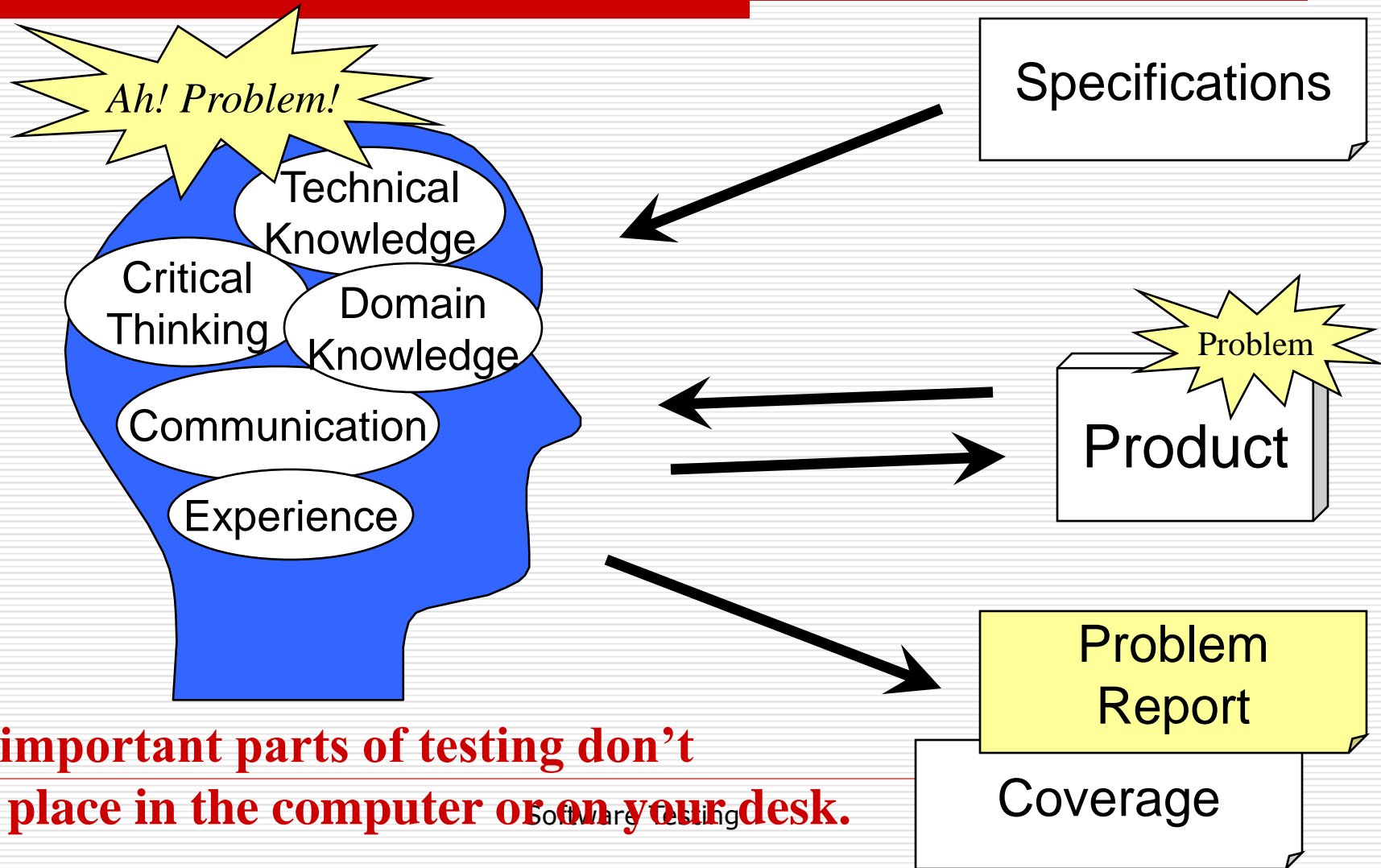
- ❑ A good tester is mischievous
- ❑ Emphasis on breadth of expertise

# Slogan !

---

**Testing is science.  
Develop your scientific mind.**

# Testing is in Your Head



# A Tester's Attitude

---

## ☐ Cautious

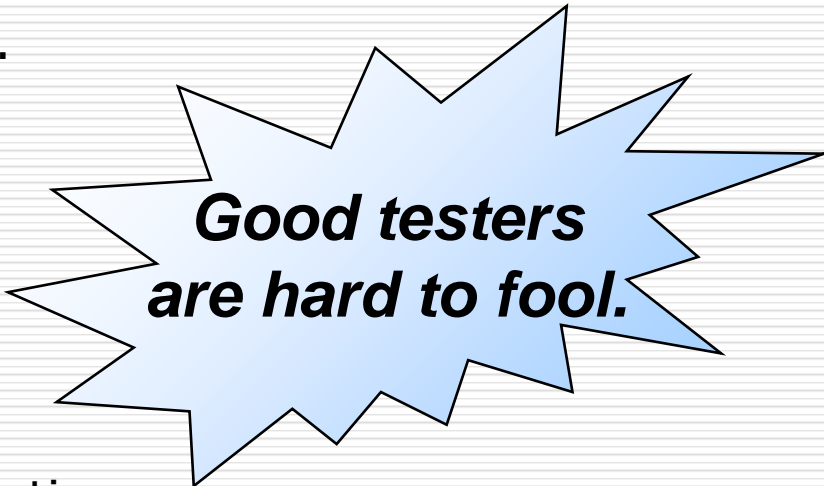
- Jump to conjectures (phỏng đoán), not conclusions (kết luận).
- Practice admitting "I don't know."
- Have someone check your work.

## ☐ Curious

- What would happen if...?
- How does that work?
- Why did that happen?

## ☐ Critical

- Proceed by conjecture and refutation
- Actively seek counter-evidence.



***Good testers  
are hard to fool.***

# Standard Testing Questions

---

- ☐ Did this test execution succeed or fail?
  - Oracles
- ☐ How shall we select test cases?
  - Selection; generation
- ☐ How do we know when we've tested enough?
  - Adequacy
- ☐ What do we know when we're done?
  - Assessment

# Software Testing

---

Classifying aspects of testing



# Software Testing categories

---

- ❑ Software Testing Levels
- ❑ Software Testing Methods
- ❑ Software Testing Types
- ❑ Software Testing Artifacts
- ❑ Software Testing Tools
- ❑ Software Testing Resources
- ❑ General

# Software Testing Levels

---

- ☐ Unit Testing
- ☐ Integration Testing
- ☐ System Testing
- ☐ Acceptance Testing
- ☐ Other Testing Levels

# Software Testing Methods

---

- ☐ Black Box Testing
- ☐ White Box Testing
- ☐ Gray Box Testing
- ☐ Manual Testing
- ☐ Automated Testing
- ☐ Other Testing Methods

# Software Testing Types

---

- ☐ Desktop Applications Testing
- ☐ Web Applications Testing
- ☐ Performance Testing
- ☐ Smoke Testing
- ☐ Installation Testing
- ☐ Look & Feel Testing
- ☐ Usability Testing
- ☐ Regression Testing
- ☐ Exploratory Testing
- ☐ Other Testing Types

# Software Testing Artifacts

---

- ☐ Test Plan
- ☐ Test Cases
- ☐ Bug Reports
- ☐ Test Reports
- ☐ Other Testing Artifacts

# Software Testing Tools

---

- ☐ Test Management Tools
- ☐ Functional Test Automation Tools
- ☐ Performance Test Automation Tools
- ☐ Bug Tracking Tools
- ☐ Other Testing Tools

# Testing tools that we use most frequently:

---

- ❑ **Automated Functional Testing:** Selenium IDE/RC, Rational Robot, Functional Tester, OTP, OC, FitNesse, TestComplete, SilkTest
- ❑ **Automated performance testing:** LoadRunner, Rational Robot, ATS
- ❑ **Secure Testing:** Web Vulnerability Scanner, WebInspect, Rational AppScan, Tenable Network Security, GUI Zenmap, Web Application Security, Paros, Wikto, Grendel-Scan, MaxPatrol
- ❑ **Accessibility Testing:** OTP, Watchfire Bobby
- ❑ **Issue Management Tools:** Atlassian, Bugzilla, Edgewall, VersionOne, Mantis, FogBugz

# Software Testing Resources

---

- ❑ Software Testing Books
- ❑ Software Testing Sites
  - <http://www.softwaretestingfundamentals.com>
  - <http://www.softwaretestinghelp.com>
  - <http://www.vietnamesetestingboard.org/>
- ❑ Software Testing Jobs



# Software Testing Certifications

---

## ☐ Vendors who offer Testing Certifications:

- Mercury
- Segue
- Rational
- Empirix

# Software Testing Certifications

---

- ❑ Vendor-Neutral Testing Certifications:
  - Certified Software Quality Analyst (CSQA)
  - Certified Software Test Engineer (CSTE)
  - ISTQB Certified Tester, Foundation Level (CTFL)
  - Quality Improvement Associate Certification (CQIA)
  - Certified Test Manager (CTM)
  - Certified Software Test Professional (CSTP)
  - Six Sigma Black Belt Certification (SSBB).

# Basic Methods of Testing

---

- ☐ Black Box
- ☐ White Box
- ☐ Gray Box

# Black Box Testing

---

- ❑ View the program as a black box
- ❑ Exhaustive testing is infeasible even for tiny programs
  - Can never guarantee correctness
  - Fundamental question is economics
  - “Maximize investment”
    - ❑ Example: “Partition test”

# Black Box Tests

---



- ❑ Targeted at the apparent simplicity of the software
  - Makes assumptions about implementation
  - Good for testing component interactions
- ❑ Tests the interfaces and behavior

# White Box Testing

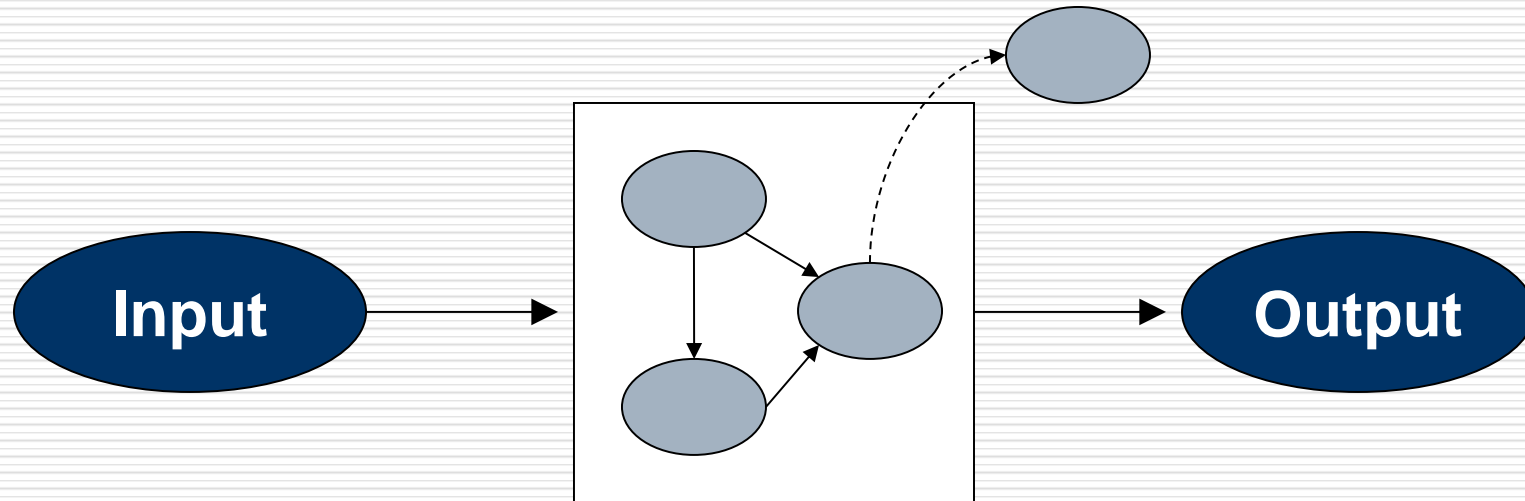
---

- ❑ Investigate the internal structure of the program
- ❑ Exhaustive path testing is infeasible
- ❑ Does not even guarantee correctness
  - Specification is needed
  - Missing paths
  - Data dependent paths

```
if (a-b < epsilon)
```

# White Box Tests

---



- ❑ Targeted at the underlying complexity of the software
  - Intimate knowledge of implementation
  - Good for testing individual functions
- ❑ Tests the implementation and design

# Black and White Box Testing Methods

---

## Black box

- Equivalence partitioning
- Boundary-value analysis
- Cause-effect graphing
- Error guessing
- State space exploration

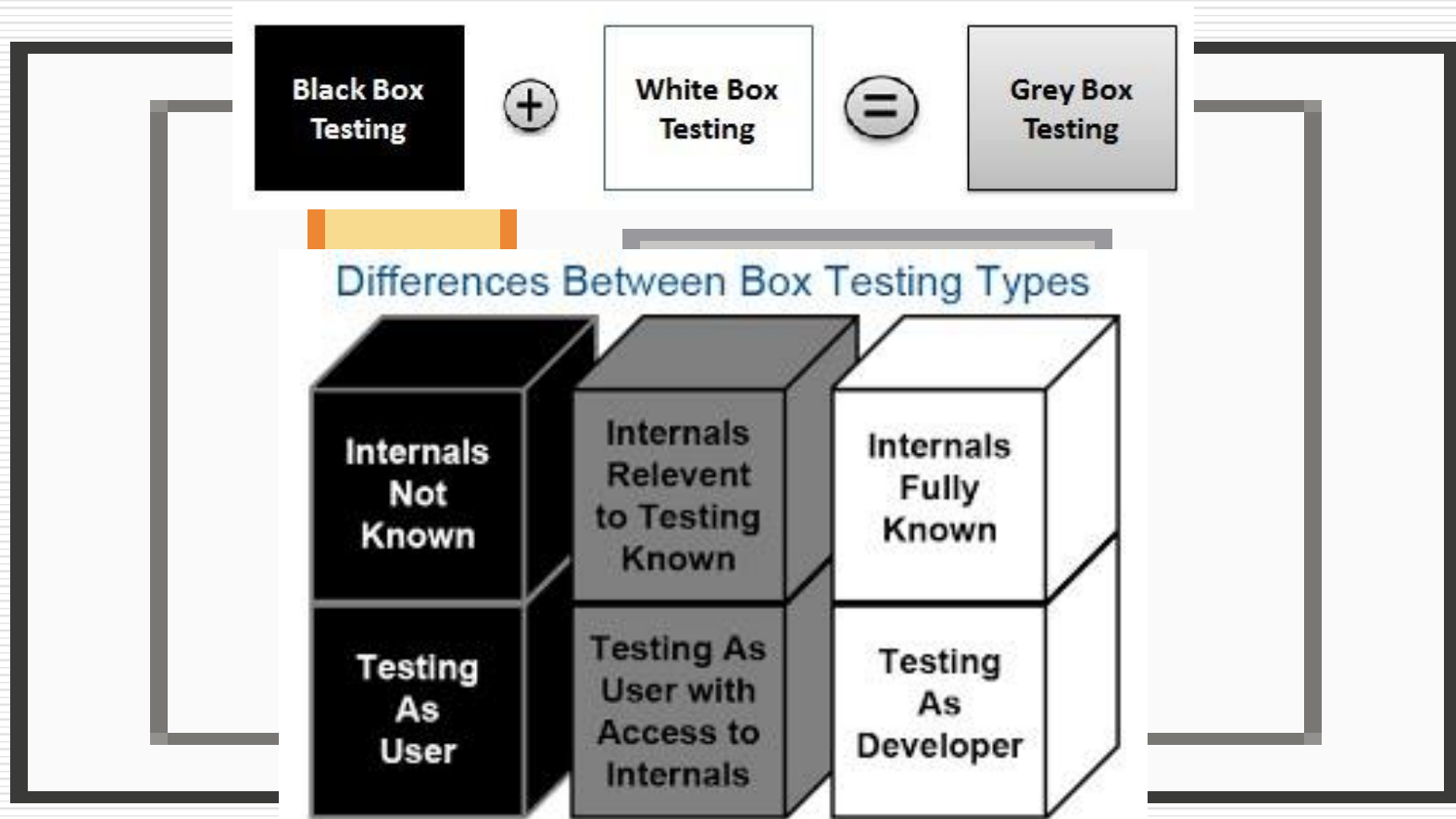
## White box

- Coverage/Adequacy
- Data flow analysis
- Cleanness
- Correctness
- Mutation
- Slicing



# Gray Box Testing

---



Which level can be applied for  
Apply? to CSP

---

# Structural Coverage Testing

---

- ❑ (In)adequacy criteria
  - If significant parts of program structure are not tested, testing is surely inadequate
- ❑ Control flow coverage criteria
  - Statement (node, basic block) coverage
  - Branch (edge) and condition coverage
  - Data flow (syntactic dependency) coverage
  - Various control-flow criteria
- ❑ Attempted compromise (dung hòa) between the impossible and the inadequate

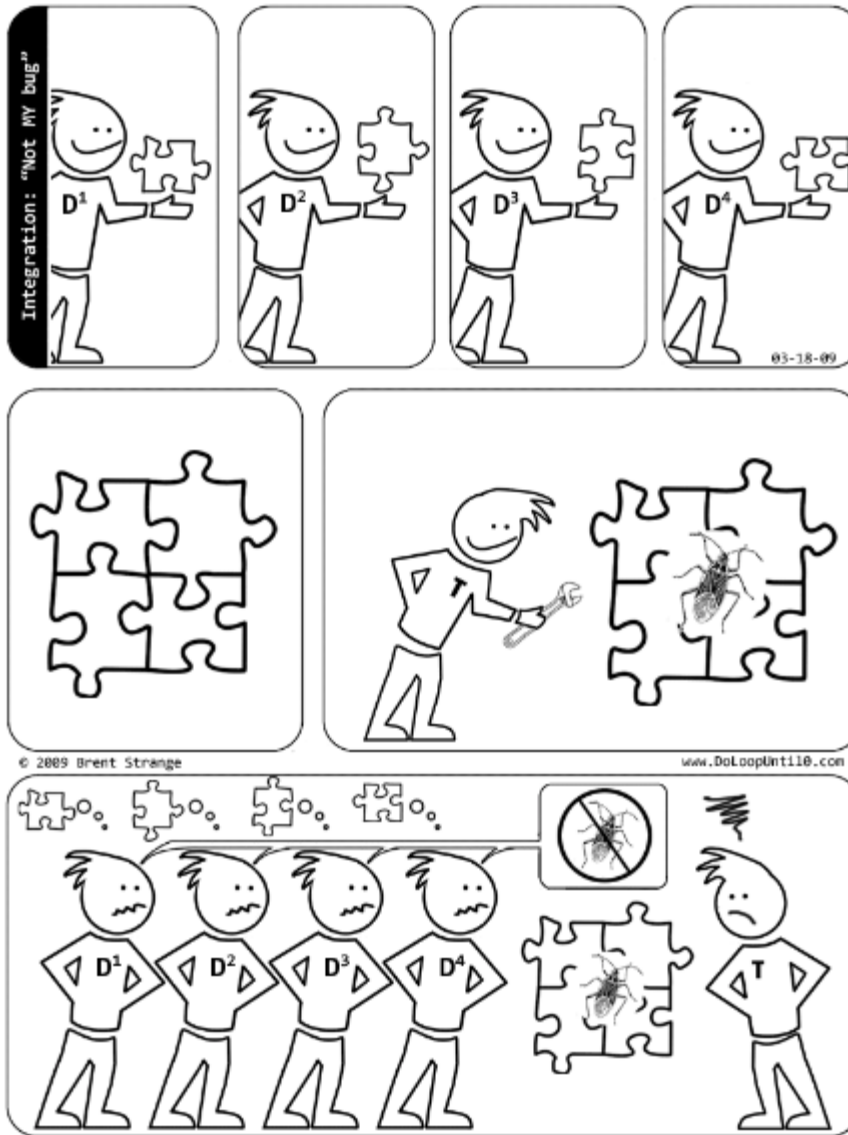
# Testing Levels

---

## Different testing levels

- Unit test (procedure boundaries)
- Module test
- Function test (discrepancies external spec.)
- System test (discrepancies with original objectives)
  - Facility test
  - Volume test
  - Usability test
  - Security test
  - Performance test
  - ...

DO  
LOOP UNTIL 0



# The Budget Coverage Criterion

---

- A common answer to “when is testing done”
  - When the money is used up
  - When the deadline is reached
- This is sometimes a rational approach!
  - Implication 1: Test selection is more important than stopping criteria.
  - Implication 2: Practical comparison of approaches must consider the cost of test case selection

# Test Cycle: What You Do With a Build

---

1. Receive the product.

- Formal builds
- Informal builds
- Save old builds.

2. Clean your system.

- Completely uninstall earlier builds.

3. Verify testability.

- Smoke testing
  - Suspend test cycle if the product is untestable.
-

# Test Cycle: What You Do With a Build

---

4. Determine what is new or changed.
    - Change log
  5. Determine what has been fixed.
    - Bug tracking system
  6. Test fixes.
    - Many fixes fail!
    - Also test nearby functionality.
  7. Test new or changed areas.
    - Exploratory testing.
-



# Test Cycle: What You Do With a Build

---

## 8. Perform regression testing.

- Not performed for an incremental cycle.
- Automated vs. manual
- Important tests first!

## 9. Report results.

- Coverage
  - Observations
  - Bug status (new, existing, reopened, closed)
  - Assessment of quality
  - Assessment of testability
-

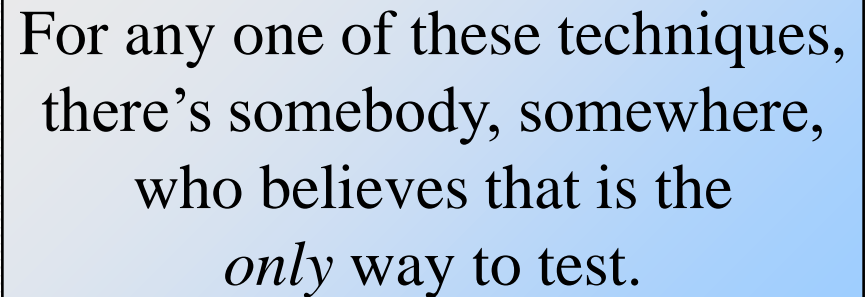
# Apply to CPS

---

# General Black Box Testing Techniques

---

- ☐ Function testing
- ☐ Domain testing
- ☐ Stress testing
- ☐ Flow testing
- ☐ Scenario testing
- ☐ User testing
- ☐ Regression testing
- ☐ Risk testing
- ☐ Claims testing
- ☐ Random Testing



For any one of these techniques,  
there's somebody, somewhere,  
who believes that is the  
*only* way to test.

# Smoke Testing

---

## □ MSDN:

- *smoke testing* describes the process of validating code changes before the changes are checked into the product's source tree.
- After code reviews, smoke testing is the most cost effective method for identifying and fixing defects in software.
- Smoke tests are designed to confirm that changes in the code function as expected and do not destabilize an entire build.

# Smoke Testing

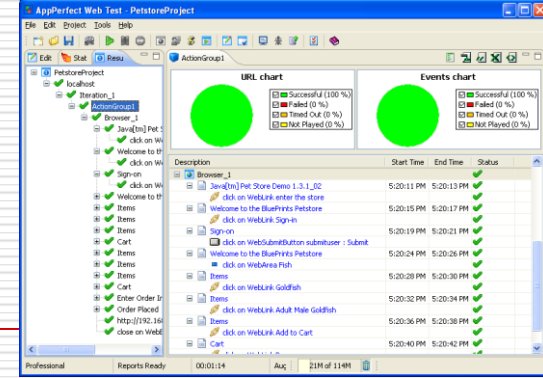
---



## □ On “What is ?”

- Smoke testing is non-exhaustive software testing, ascertaining that the most crucial functions of a program work, but not bothering with finer details.
- The term comes to software testing from a similarly basic type of hardware testing, in which the device passed the test if it didn't **catch fire** the first time it was **turned on**

# Function Testing



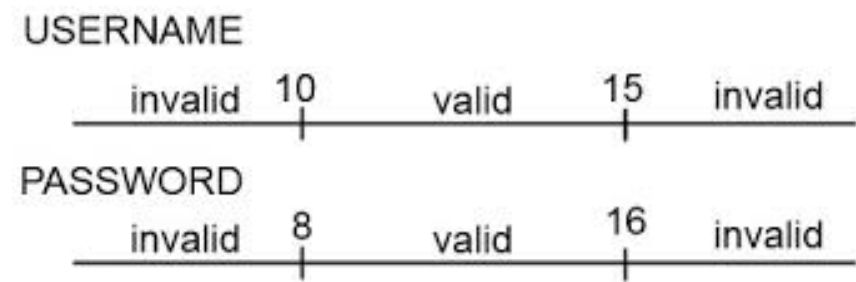
**Key Idea:** *“March through the functions”*

**Summary:**

- A function is something the product can do.
- Identify each function and sub-function.
- Determine how you would know if they worked.
- Test each function, one at a time.

**Good for:** assessing capability rather than reliability  
(khả năng và độ tin cậy)

# Domain Testing



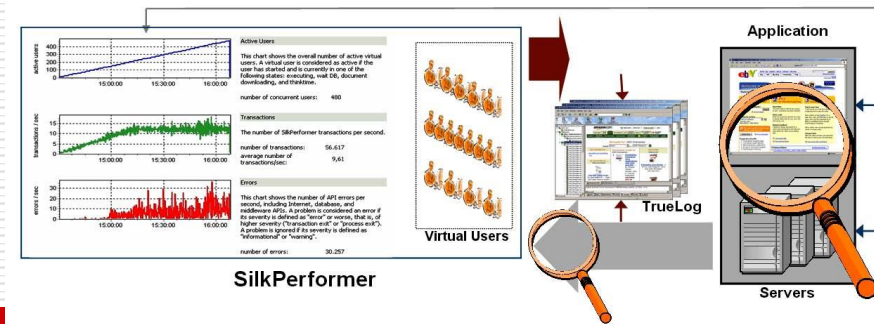
**Key Idea:** *“Divide the data, and conquer”*

**Summary:**

- A domain is a set of test data.
- Identify each domain of input and output.
- Analyze limits and properties of each domain.
- Identify combinations of domains to test.
- Select coverage strategy:  
    *e.g., exhaustive, boundaries, best representative*

**Good for:** all purposes

# Stress Testing



**Key Idea:** *“Overwhelm the product”*

**Summary:**

- Select test items and functions to stress.
- Identify data and platform elements that relate to them.
- Select or generate challenging data and platform configurations to test with:  
*e.g., large or complex data structures, high loads, long test runs, many test cases*

**Good for:** performance, reliability, and efficiency assessment



Difference between Performance Testing, Load Testing and Stress Testing.



# Flow Testing

---

**Key Idea:** *“Do one thing after another”*



**Summary:**

- Define test procedures or high level cases that incorporate many events and states.
- The events may be in series, parallel, or some combination thereof.
- Don't reset the system between events.

**Good for:** finding problems fast  
(however, bug analysis is more difficult)

---

# Scenario Testing

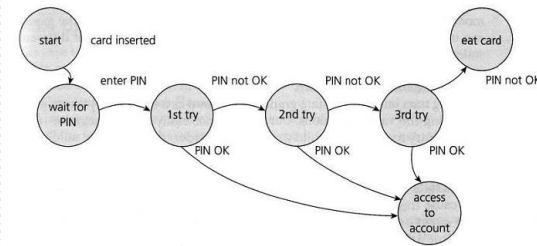
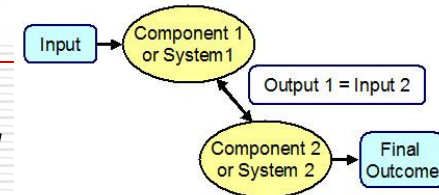


FIGURE 4.2 State diagram for PIN entry



[www.softwaretestinggenius.com](http://www.softwaretestinggenius.com)

**Key Idea:** *“Test to a compelling story”*

**Summary:**

- Design tests that involve meaningful and complex interactions with the product.
- A good scenario test is a plausible story of how someone who matters might do something that matters with the product.
- Incorporate multiple elements of the product, and data that is realistically complex.

**Good for:** Finding problems that seem important

# User Testing

---

**Key Idea:** *“Involve the users”*

**Summary:**

- Identify categories of users.
- Understand favored and disfavored users.
- Find these users and have them do testing or help you design tests.
- User testing is powerful when you involve a variety of users.



**Good for:** all purposes

---

# Regression Testing

Regression:  
"when you fix one bug, you  
introduce several newer bugs."

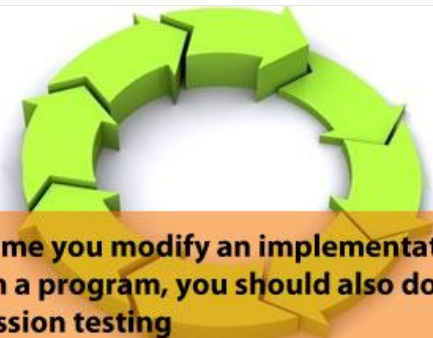


**Key Idea:** *"Test the changes"*

**Summary:**

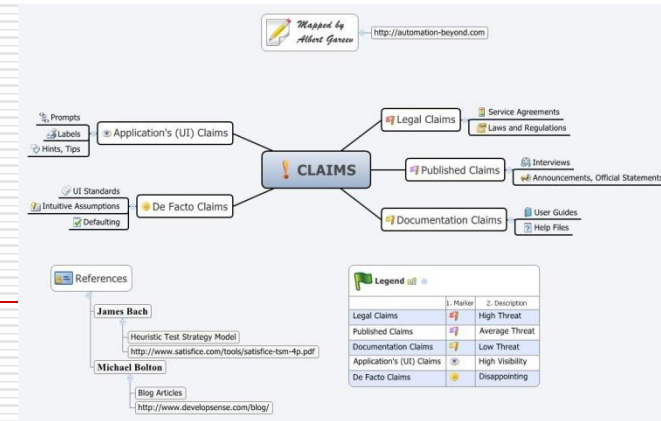
- Identify what product elements changed.
- Identify what elements could have been impacted by the changes.
- Coverage strategy:  
e.g., *recent bug fixes, past bug fixes, likely elements, all elements*

**Good for:** managing risks related to product enhancement



Any time you modify an implementation within a program, you should also do regression testing

# Claims Testing



**Key Idea:** *“Verify every claim”*

**Summary:**

- Identify specifications (implicit or explicit).
- Analyze individual claims about the product.
- Ask customer to clarify vague claims.
- Verify each claim.
- Expect the specification and product to be brought into alignment.

**Good for:** Simultaneously testing the product and specification, while refining expectations

# Risk Testing

---



**Key Idea:** *“Imagine a problem, then look for it.”*

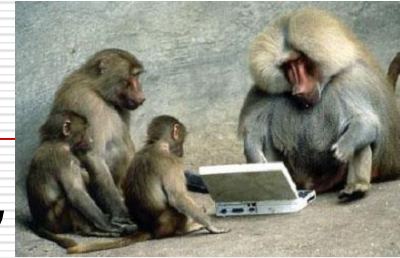
**Summary:**

- What kinds of problems could the product have?
- How would you detect them if they were there?
- Make a list of interesting problems and design tests specifically to reveal them.
- It may help to consult experts, design documentation, past bug reports, or apply risk heuristics.

**Good for:** making best use of testing resources;  
leveraging experience

---

# Random Testing



**Key Idea:** *“Run a million different tests”*

- Summary:**
- Look for an opportunity to automatically generate thousands of slightly different tests.
  - Create an automated, high speed evaluation strategy.
  - Write a program to generate, execute, and evaluate all the tests.



*Development of tests using a black box method, in which test cases are chosen to match the functional cross-section, usually using an algorithm of pseudo-random selection.*

**Good for:** Assessing reliability across input and time.



# General White Box Testing Techniques

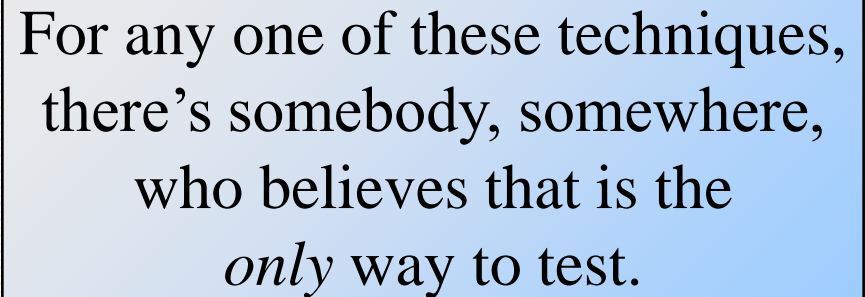
---

## ☐ Coverage Testing

- Branch
- Logic
- Statement
- Data flow
- ...

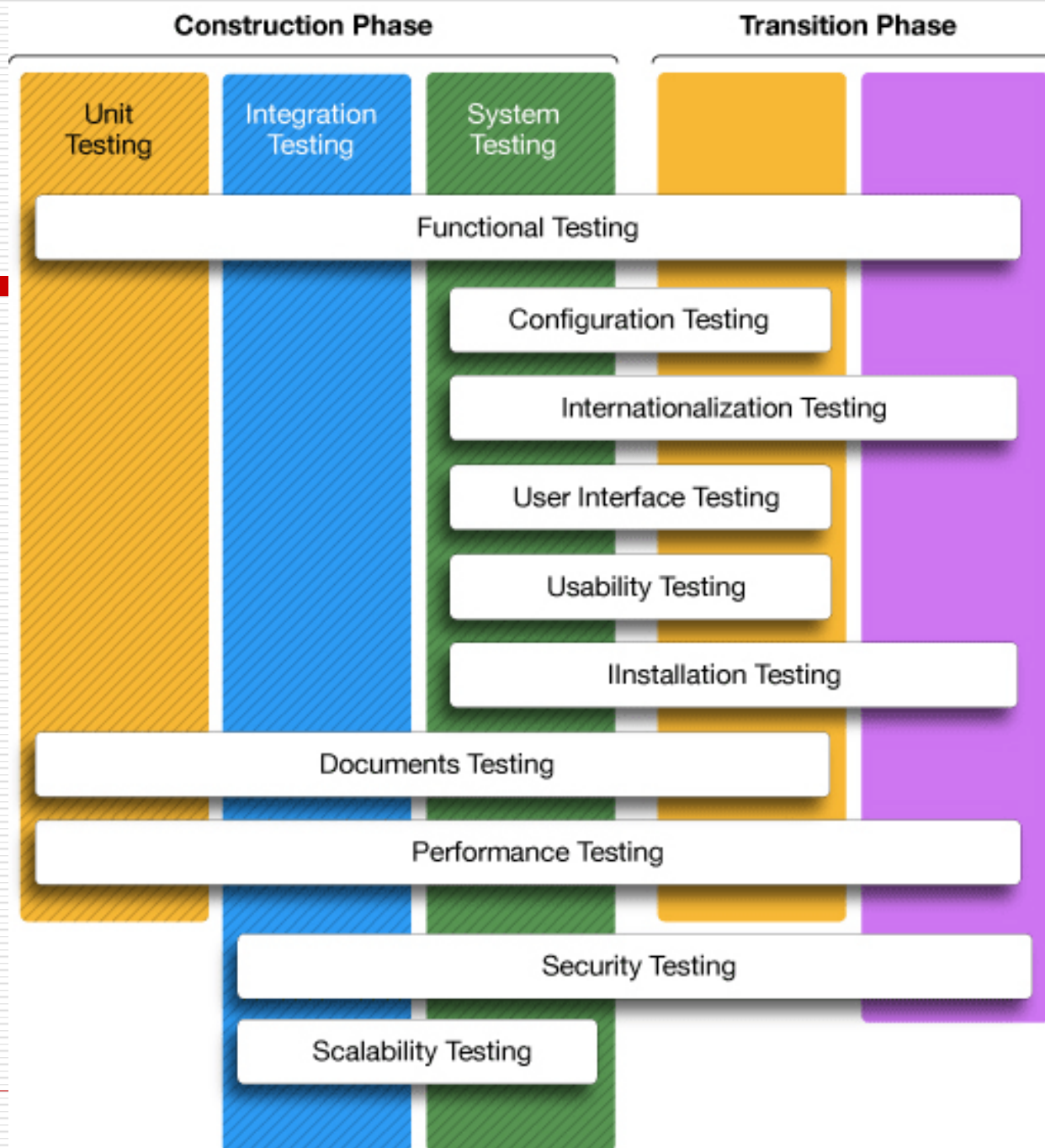
## ☐ Unit Testing

- DotNet Unit
- PhP Unit
- Cpp Unit
- Junit
- Ruby Unit
- ....



For any one of these techniques,  
there's somebody, somewhere,  
who believes that is the  
*only* way to test.





# Manual Vs Automation Testing

