# Introduction to Software Testing

# Chapter 6
# Practical Considerations

**Paul Ammann & Jeff Offutt**

Updated by Sunae Shin

## Introduction

- Chapters 1-5 fill up a "toolbox" with useful criteria for testing software
- Topics :
  - Regression testing
  - Integrating software components and testing
  - Integrating testing with development
  - Test plans
- Most importantly :
  - In any activity, knowing the tools is only the first step
  - The key is utilizing the tools in effective ways

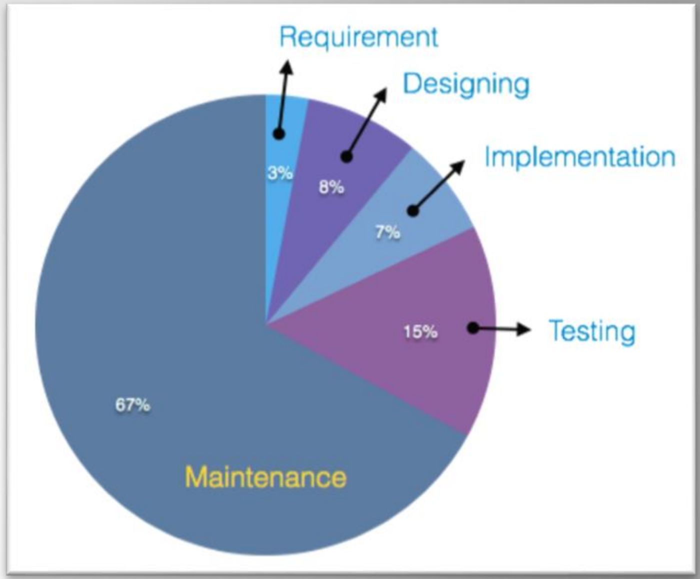
# Chapter 6 Outline

## Test Process

**We know <u>what</u> to do … but now … <u>how</u> can we do it?**

# Changes in Software Production

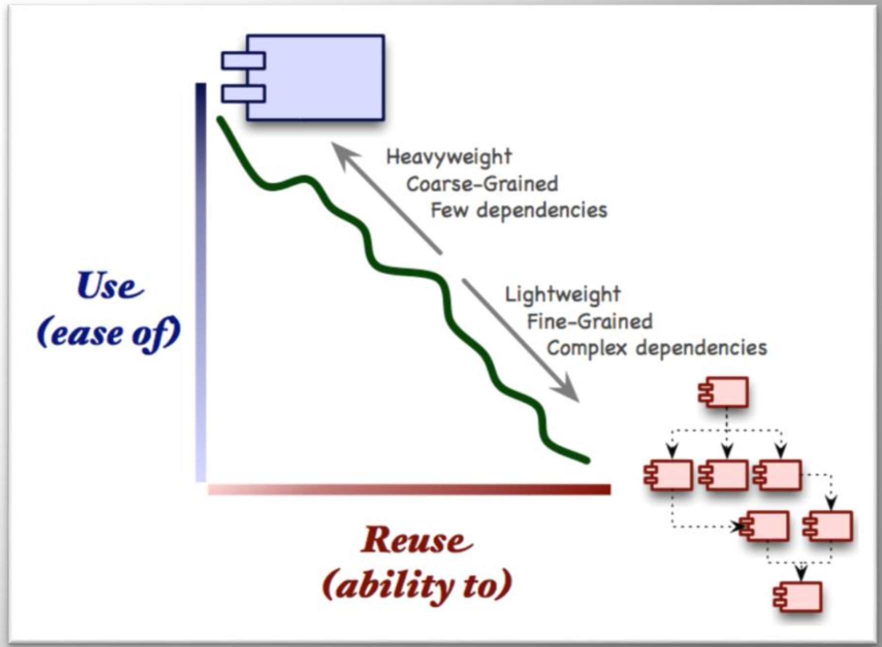
- **We do more maintenance than construction**
  - **Postdelivery maintenance**
    - **Any change to any component of the product (including documentation) after it has passed the acceptance test**



- **Modularity**
  - **When a large product consists of a single monolithic block of code**
    - **Maintenance is a nightmare**
    - **Even for the author, attempting to debug the code is extremely difficult**
  - **Solution**
    - **Break the product into smaller pieces, called modules**

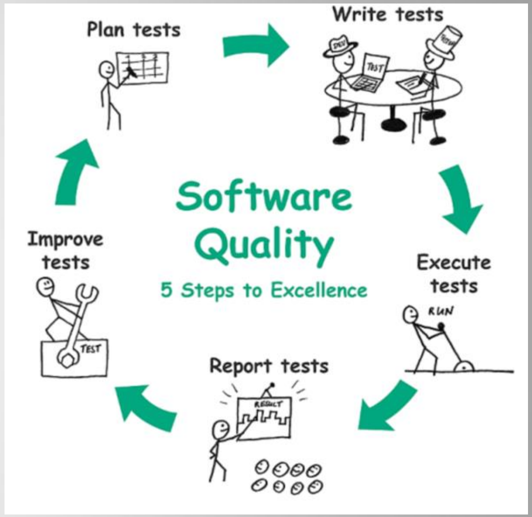
## Changes in Software Production

- **We are reusing code in many ways**
  - ***Reuse* refers to using components of one product to facilitate the development of a different product with a different functionality**
  - **Reusable components**
    - **Class**
    - **Code fragment**
    - **Design**
    - **Part of a manual**
    - **Set of test data, a contract**
    - **Duration and cost estimate**



# Changes in Software Production

- **Level 4 thinking (Testing Levels Based on Test Process Maturity) requires the recognition that quality is usually more crucial than efficiency**
  - Requires that programmers respect testers



- **Quality management**
  - Also called software quality assurance (SQA)
  - Serves as an umbrella activity that is applied throughout the software process
  - Involves doing the software development correctly versus doing it over again
  - Reduces the amount of rework, which results in lower costs and improved time to market

# Changes in Software Production

- **Software Quality Assurance (SQA) group**
  - **Assists the software team in achieving a high-quality product**
  - **Views the software from the <u>customer's point of view</u>**
    - **Does the software adequately meet quality factors?**
    - **Has software development been conducted according to pre-established standards?**
  - **Performs a set of of activities that address quality assurance planning, oversight, record keeping, analysis, and reporting**
    - **<u>Prepares</u> an SQA plan for a project**
    - **<u>Participates</u> in the development of the project's software process description**
    - **<u>Reviews</u> software engineering activities to <u>verify</u> compliance with the defined software process**
    - **Helps to <u>collect</u> and <u>analyze</u> software metrics**

# Test Activities

| | |
|---|---|
| Software requirements | Define test objectives (criteria)<br>Project test plan |
| System design | Design system tests<br>Design acceptance tests<br>Design usability test, if appropriate |
| Intermediate design | Specify system tests<br>Integration and unit test plans<br>Acquire test support tools<br>Determine class integration order |
| Detailed design | Create tests or test specifications |

# Test Activities (2)

| | |
|---|---|
| Implementation | Create tests<br>Run tests when units are ready |
| Integration | Run integration tests |
| System deployment | Apply system test<br>Apply acceptance tests<br>Apply usability tests |
| Operation and maintenance | Capture user problems<br>Perform regression testing |

# Test Activities - details

- **Requirements Analysis and Specification**

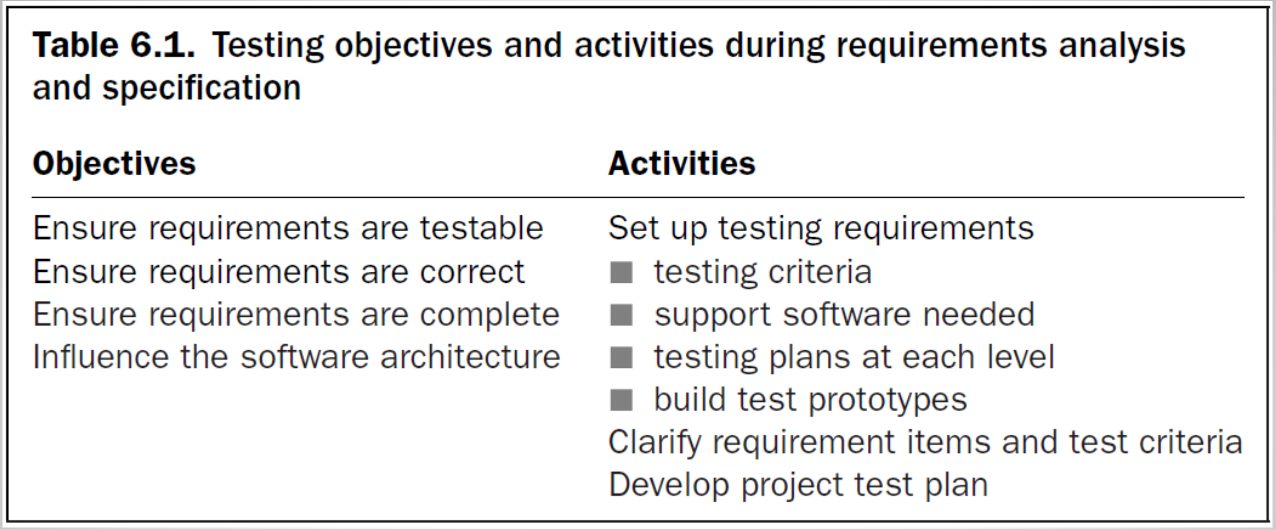


**Table 6.1.** Testing objectives and activities during requirements analysis and specification

| Objectives | Activities |
|---|---|
| Ensure requirements are testable | Set up testing requirements |
| Ensure requirements are correct | ■ testing criteria |
| Ensure requirements are complete | ■ support software needed |
| Influence the software architecture | ■ testing plans at each level |
| | ■ build test prototypes |
| | Clarify requirement items and test criteria |
| | Develop project test plan |

- **System Design**

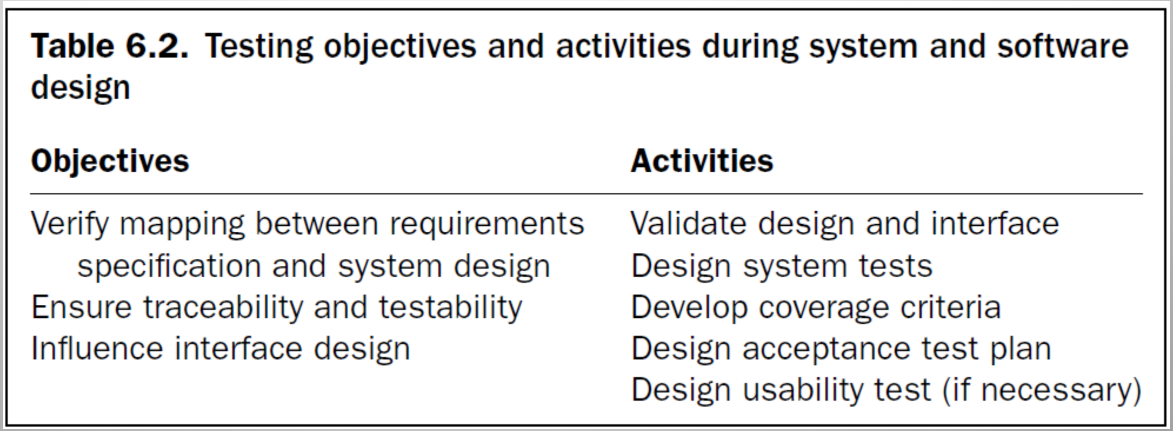


**Table 6.2.** Testing objectives and activities during system and software design

| Objectives | Activities |
|---|---|
| Verify mapping between requirements specification and system design | Validate design and interface |
| | Design system tests |
| Ensure traceability and testability | Develop coverage criteria |
| Influence interface design | Design acceptance test plan |
| | Design usability test (if necessary) |

# Test Activities - details

- **Intermediate Design**

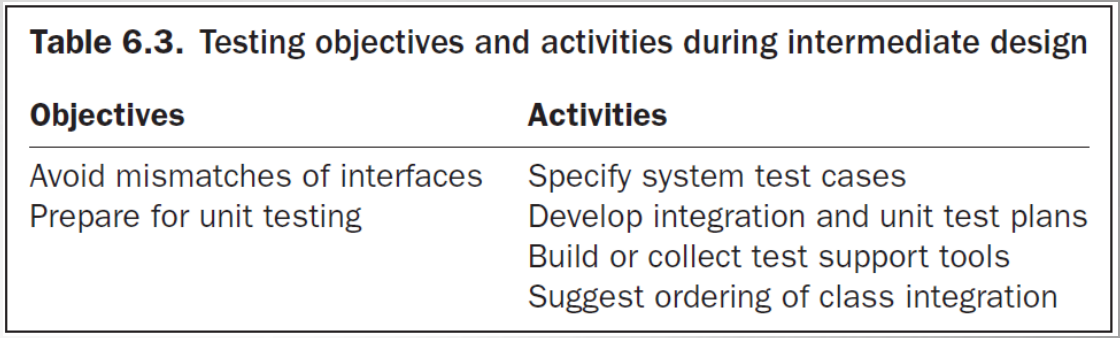


Table 6.3. Testing objectives and activities during intermediate design

| Objectives | Activities |
|---|---|
| Avoid mismatches of interfaces | Specify system test cases |
| Prepare for unit testing | Develop integration and unit test plans |
| | Build or collect test support tools |
| | Suggest ordering of class integration |

- **Detailed Design**

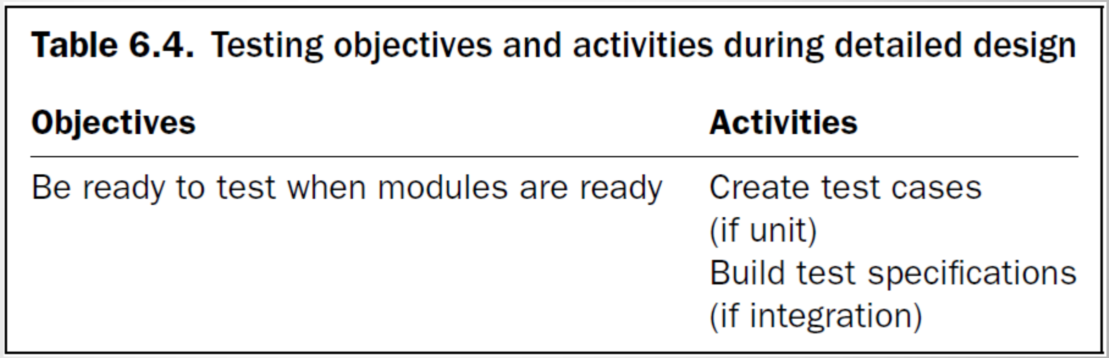


Table 6.4. Testing objectives and activities during detailed design

| Objectives | Activities |
|---|---|
| Be ready to test when modules are ready | Create test cases |
| | (if unit) |
| | Build test specifications |
| | (if integration) |

- **Implementation**

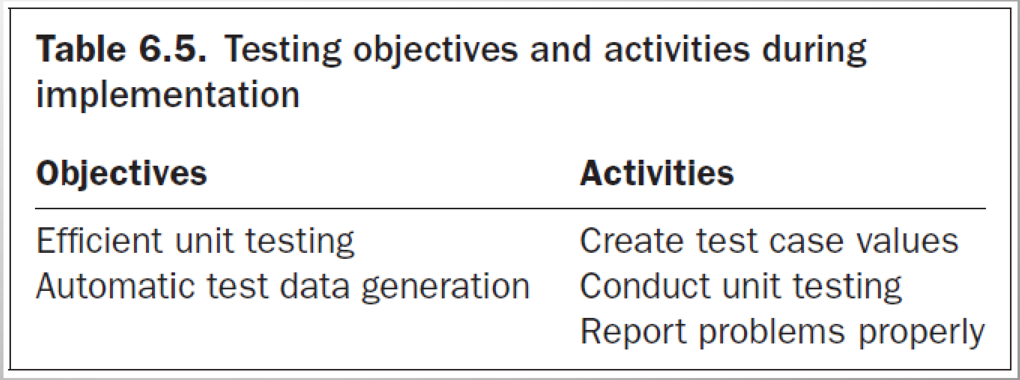


Table 6.5. Testing objectives and activities during implementation

| Objectives | Activities |
|---|---|
| Efficient unit testing | Create test case values |
| Automatic test data generation | Conduct unit testing |
| | Report problems properly |

# Test Activities - details

- **Integration**

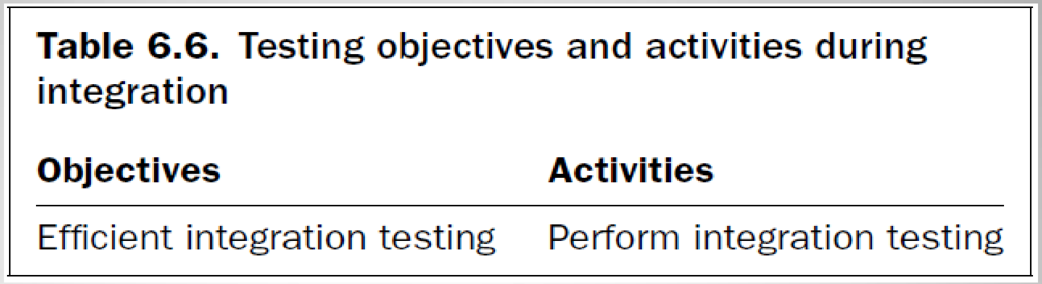


**Table 6.6.** Testing objectives and activities during integration

| Objectives | Activities |
|---|---|
| Efficient integration testing | Perform integration testing |

- **Deployment**

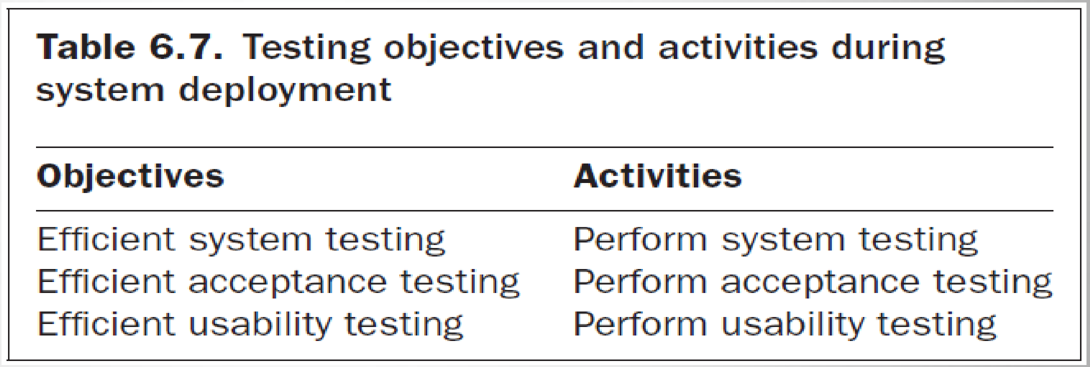


**Table 6.7.** Testing objectives and activities during system deployment

| Objectives | Activities |
|---|---|
| Efficient system testing | Perform system testing |
| Efficient acceptance testing | Perform acceptance testing |
| Efficient usability testing | Perform usability testing |

- **Maintenance**

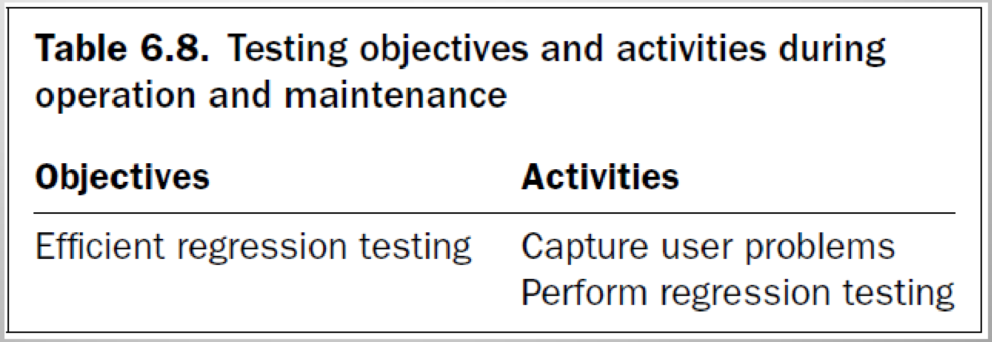


**Table 6.8.** Testing objectives and activities during operation and maintenance

| Objectives | Activities |
|---|---|
| Efficient regression testing | Capture user problems<br>Perform regression testing |

## Managing Test Artifacts

- Keep track of :
  - Test design documents
  - Tests
  - Test results
  - Automated support
- Keep track of source of tests – when the source changes, the tests must also change

## Professional Ethics

- Put quality first : Even if you lose the argument, you will gain respect
- Decouple
  - Designs should be independent of language
  - Couplings are weaknesses in the software!
- Begin test activities early

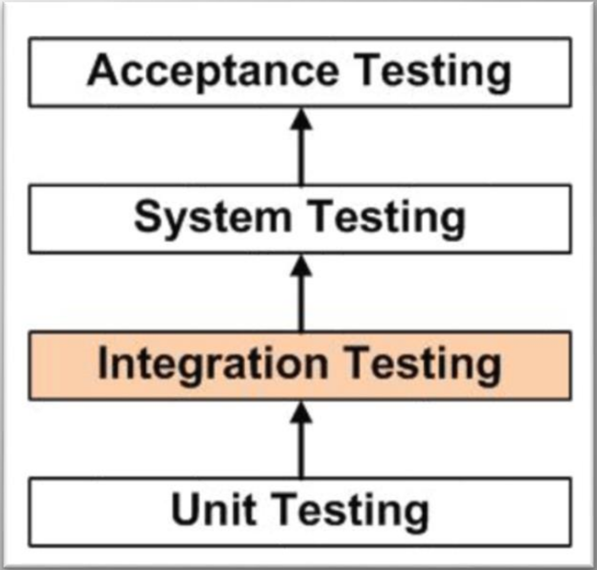
# Chapter 6 Outline

# Integration and Testing

**Big Bang Integration**

**Throw all the classes together, compile the whole program, and system test it**

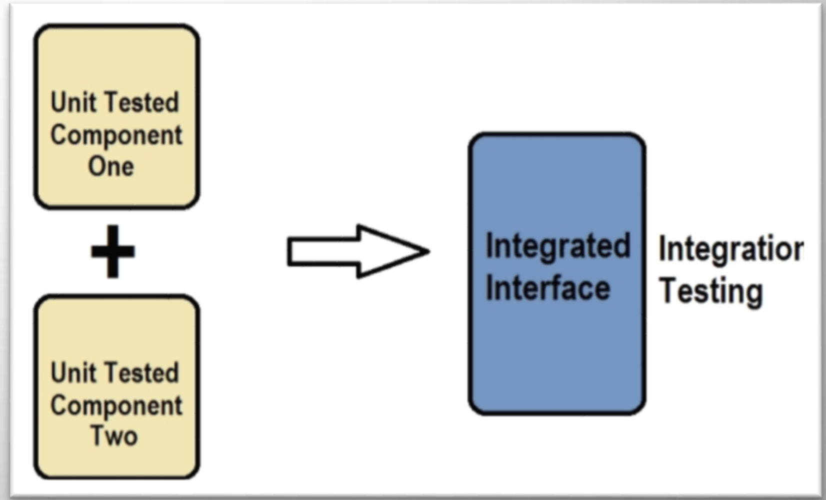
- **The polite word for this is risky**
- **The usual method is to start small, with a few classes that have been tested thoroughly**
  - **Add a small number of new classes**
  - **Test the connections between the new classes and pre-integrated classes**



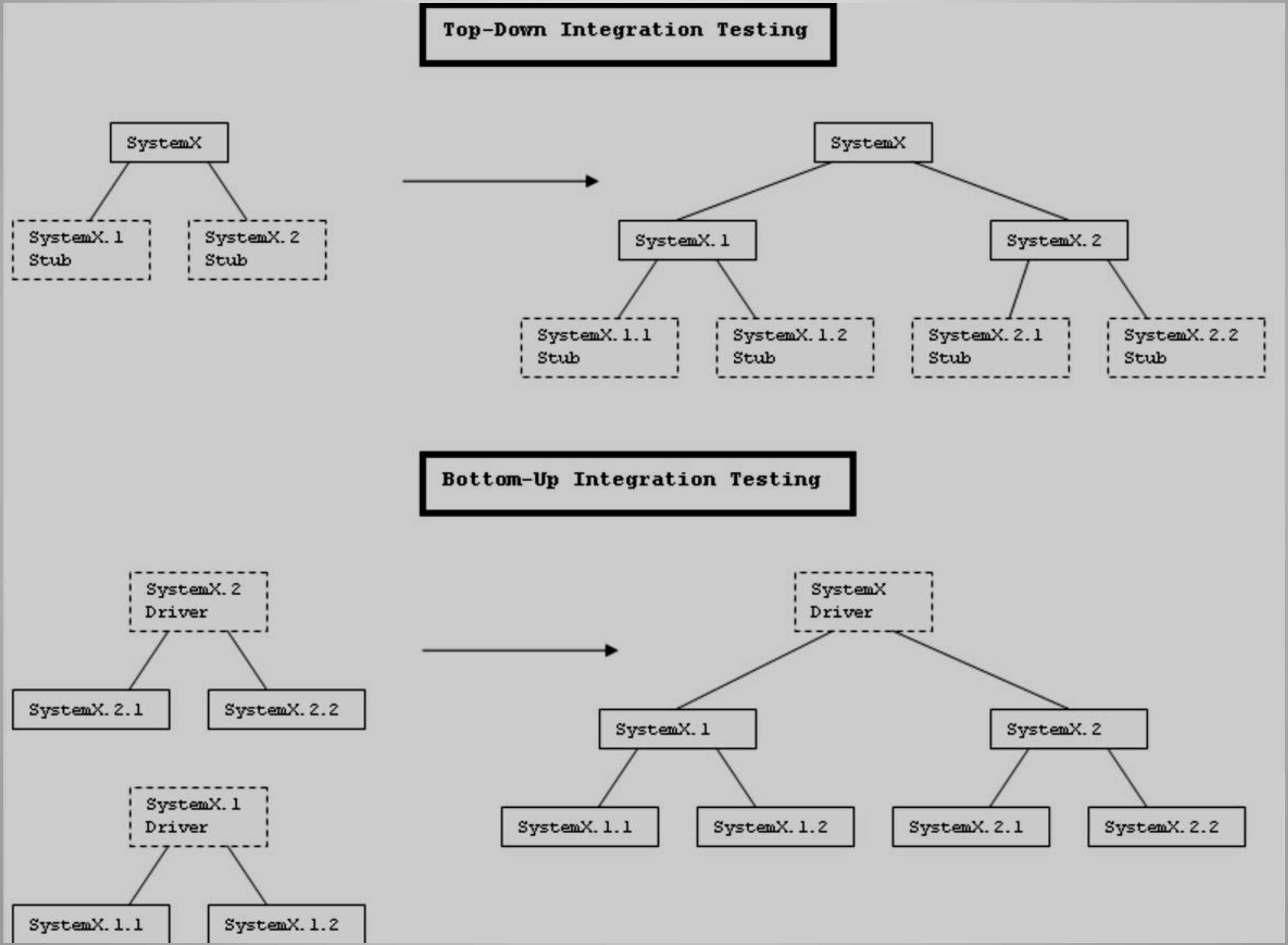
# Integration and Testing

- ***A component is a piece of a program that can be tested independently***

- ***Integration testing*** **: testing interfaces between correctly working components**
  - **Should have already been tested in isolation (unit testing)**

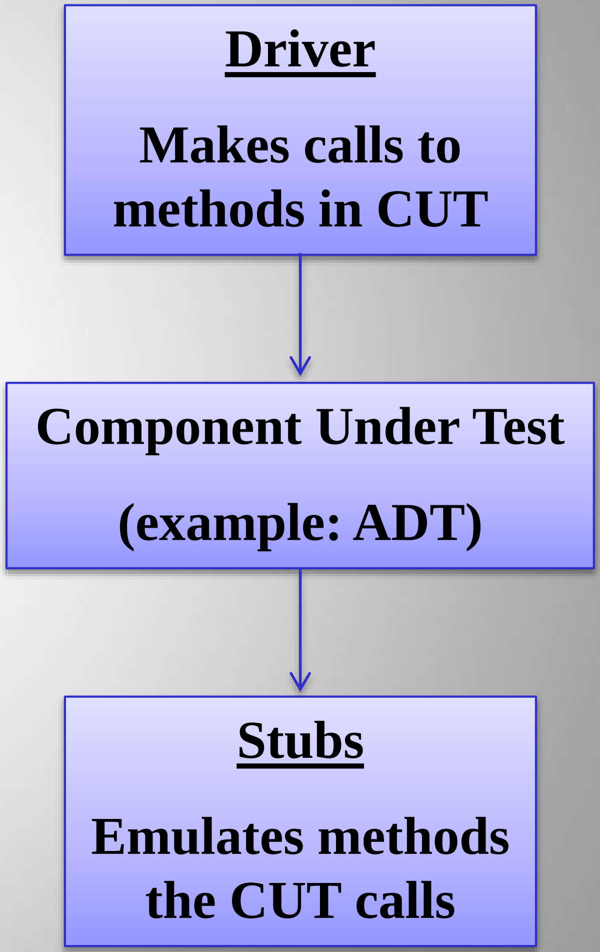


# Integration Testing



## Software Scaffolding

- When testing incomplete portions of software, testers often need extra software components – called scaffolding
  - *Scaffolding* is extra software components that are created to support integration and testing
  - Two common types of scaffolding
    - A *stub* emulates the results of a call to a method that has not been implemented or integrated yet
    - A *driver* emulates a method that makes calls to a component that is being tested



## Stubs

- The first responsibility of a stub is to allow the CUT to be compiled and linked without error
  - The signature must match
  - What if the called method <u>needs to return values</u> ?
  - These values will not be the same the full method would return
  - It may be important for testing that they satisfy certain limited constraints

- Approaches:
  1. Return constant values from the stub
  2. Return random values
  3. Return values from a table lookup
  4. Return values entered by the tester during execution
  5. Processing formal specifications of the stubbed method

## Drivers

- **Many good programmers add drivers to every class as a matter of habit**
  - **Instantiate objects and carry out simple testing**
  - **Criteria from previous chapters can be implemented in drivers**

- **Test drivers can easily be created automatically**

# Class Integration and Test Order (CITO)

- **Old programs tended to be very hierarchical**
  - **Which order to integrate was pretty easy:**
    - **Test the "leaves" of the call tree**
    - **Integrate up to the root**
    - **Goal is to minimize the number of stubs needed**

- **OO programs make this more complicated**
  - **Lots of kinds of dependencies (call, inheritance, use, aggregation)**
  - **Circular dependencies : A inherits from B, B uses C, C aggregates A**

- **CITO : *Which order should we integrate and test* ?**
  - **Must "break cycles"**
  - **Common goal : least extra work (primarily creating stubs)**

# Chapter 6 Outline

1. Regression Testing
2. Integration and Testing
3. Test Process
4. Test Plans

## Different Types of Maintenance

1. **Corrective maintenance**
2. **Perfective maintenance**
3. **Adaptive maintenance**

## Maintenance (contd)

- Corrective maintenance
  - To correct residual faults
    - Analysis, design, implementation, documentation, or any other type of faults

## Maintenance (contd)

- **Perfective maintenance**
  - **Client requests changes to improve product effectiveness**
    - **Add additional functionality**
    - **Make product run faster**
    - **Improve maintainability**

## Maintenance (contd)

- **Adaptive maintenance**
  - **Responses to changes in the *environment* in which the product operates**
    - **The product is ported to a new compiler, operating system, and/or hardware**
    - **A change to the tax code**
    - **9-digit ZIP codes**

# Regression Testing

- **Most software today has very little new development**
  - **Correcting, perfecting, adapting, or preventing problems with existing software**
  - **Composing new programs from existing components**
  - **Applying existing software to new situations**

- **Because of the deep interconnections among software components,**

  **<u>changes in one method can cause problems in methods that seem to be unrelated</u>**

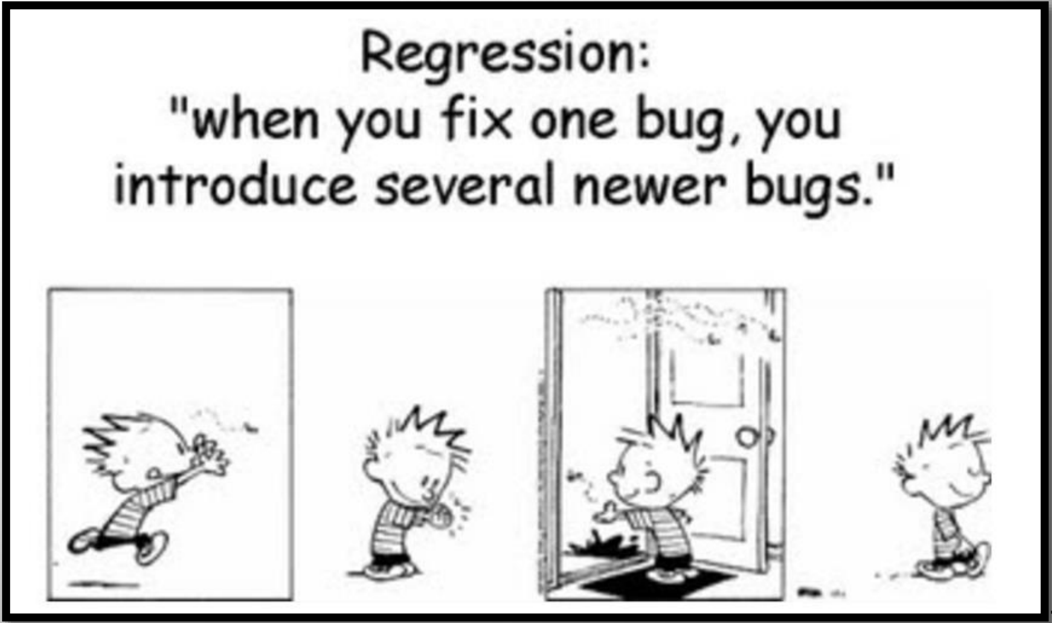
## Regression Testing

- **Regression testing is initiated when programmer fix any bug or add new code for new functionality to the system.**

**<u>Definition</u>**

**The process of re-testing software that has been modified**

- **Not surprisingly, most of our testing effort is regression testing**



# Automation and Tool Support

**Regression tests must be automated**

- **Too many tests to be run by hand**
- **Tests must be run and evaluated quickly**
  - **Often overnight, or more frequently for web applications**

- **Types of tools :**
  - **Capture / Replay – *Capture* values entered into a GUI and *replay* those values on new versions**
  - **Version control – Keeps track of collections of *tests*, expected *results*, where the tests *came from*, the *criterion* used, and their past *effectiveness***
  - **Scripting software – Manages the process of obtaining test *inputs*, *executing* the software, obtaining the *outputs*, *comparing* the results, and generating *test reports***
- **Tools are plentiful and inexpensive (often free)**

# Managing Tests in a Regression Suite

- Test suites accumulate new tests over time
- Test suites are usually run in a fixed, short, period of time
  - Often overnight, sometimes more frequently, sometimes less
- At some point, the number of tests can become unmanageable
  - We cannot finish running the tests in the time allotted
- But is it worth it?
- How many of these tests really need to be run ?

## Policies for Updating Test Suites

- Which tests to keep can be based on several policies
  - Add a new test for every problem report
  - Ensure that a coverage criterion is always satisfied

- Possible ways to choose tests to remove
  - Remove tests that do not contribute to satisfying coverage
  - Remove tests that have never found a fault (risky !)
  - Remove tests that have found the same fault as other tests (also risky !)

## Summary of Regression Testing

- **We spend far more time on regression testing than on testing new software**
- **If tests are based on covering criteria, all problems are much simpler**
  - **We know why each test was created**
  - **We can make rationale decisions about whether to run each test**
  - **We know when to delete the test**
  - **We know when to modify the test**
- **Automating regression testing will save much more than it will cost**

# Chapter 6 Outline

1. Regression Testing
2. Integration and Testing
3. Test Process
4. Test Plans

## Test Plans

- **The most common question I hear about testing is**

  **“ *How do I write a test plan*? ”**

  - **Good testing is more important than proper documentation**
  - **However – documentation of testing can be very helpful**

- **Most organizations have a list of topics, outlines, or templates**

## Standard Test Plan

- **ANSI / IEEE Standard 829-1983 is ancient but still used**

**<u>Test Plan</u>**

**A document describing the scope, approach, resources, and schedule of intended testing activities.**

**It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning.**

- **Many organizations are required to adhere to this standard**

# Types of Test Plans

- **Mission plan – tells "why"**
  - Usually one mission plan per organization or group
  - Least detailed type of test plan

- **Strategic plan – tells "what" and "when"**
  - Usually one per organization, or perhaps for each type of project
  - General requirements for coverage criteria to use

- **Tactical plan – tells "how" and "who"**
  - One per product
  - More detailed
  - Living document, containing test requirements, tools, results and issues such as integration order

# Test Plan Contents – System Testing

- Purpose
- Target audience and application
- Deliverables
- Information included
  - Introduction
  - Test items
  - Features tested
  - Features not tested
  - Test criteria
  - Pass / fail standards
  - Criteria for starting testing
  - Criteria for suspending testing
  - Requirements for testing restart
  - Hardware and software requirements
  - Responsibilities for severity ratings
  - Staffing & training needs
  - Test schedules
  - Risks and contingencies
  - Approvals