Testing Techniques

CHAPTER 8, SOFTWARE ENGINEERING 10TH ED., I. SOMMERVILLE

LECTURE NOTES PREPARED BY

ASST. PROF. MELODY ANGELIQUE RIVERA

FACULTY, COLLEGE OF COMPUTER STUDIES, SILLIMAN UNIVERSITY

“Testing can only show the presence of errors, not their absence”

-Edsger Dijkstra, 1972
(an early contributor to the development of software engineering)

Why test your software? (1)

- **To show that a program does what it is intended to do**
  - Demonstrate to the developer and the customer that the software meets its requirements
  - For custom software, there should be at least one test for every requirement in the requirements document
  - For generic software products, it means that there should be tests for all of the system features that will be included in the product release
  - You may also test combinations of features to check for unwanted interactions between them

Why test your software? (2)

- **To discover program defects before it is put into use**
  - Find inputs or input sequences where the behavior of the software is incorrect, undesirable, or does not conform to its specification
    - caused by defects (bugs) in the software
  - When you test software to find defects, you are trying to root out undesirable system behavior such as
    - system crashes
    - unwanted interactions with other systems
    - incorrect computations
    - data corruption

Verification and Validation (V & V)

- **Validation**
  - "Are we building the right product?"
  - concerned with checking that the software delivers the functionality expected by the people paying for the software
- **Verification**
  - "Are we building the product right?"
  - Concerned with checking that software being developed meets its specification

Testing is part of software verification and validation

Software inspection and reviews versus Software Testing

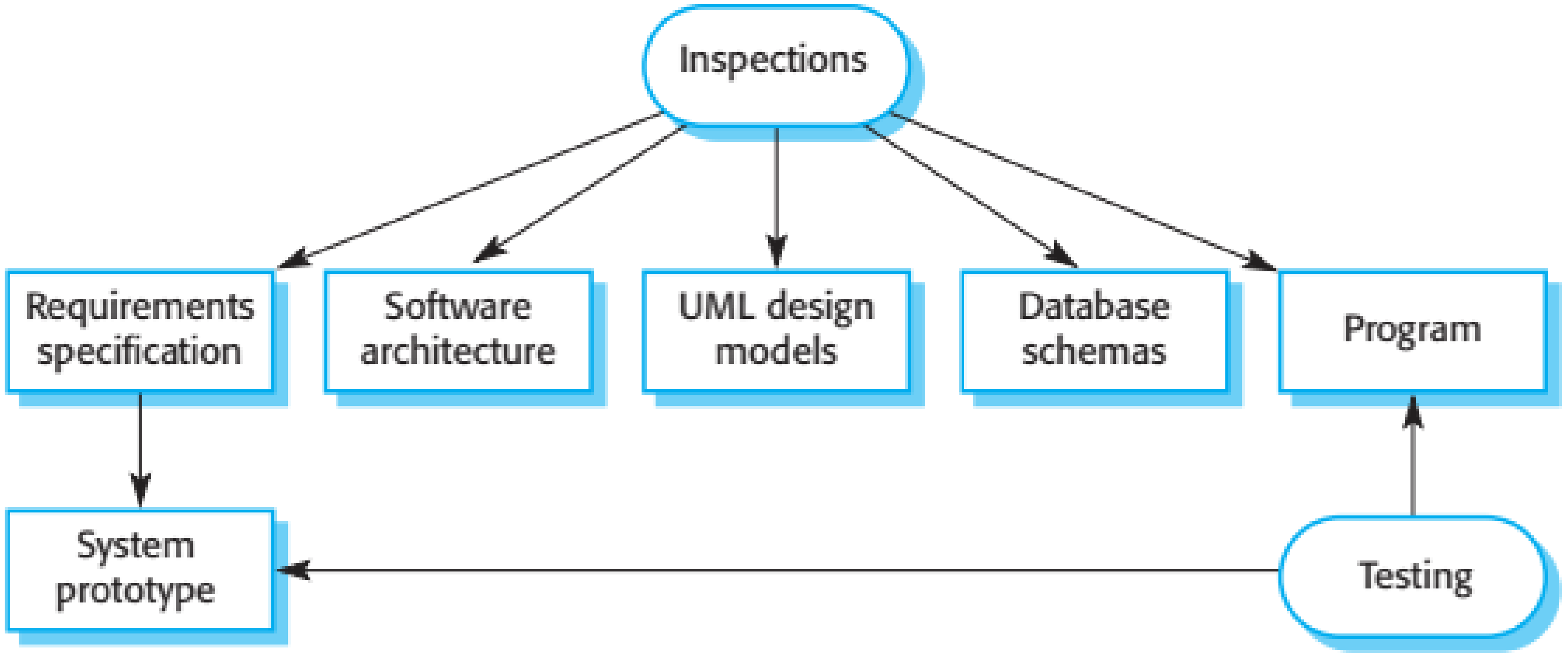
Software Inspection and Review

- No need to execute the software to verify it
- Analyzes and checks
  - System requirements
  - Design models
  - Program source code
  - Proposed system tests

Software Testing

- Needs to execute the software
- Uses test cases
- Need test data (inputs that have been devised to test a system

How inspections and testing support V & V at different stages in the software process



Advantages of software inspection (1)

- **A single inspection session can discover many errors in a system**
  - During testing, errors can hide other errors
  - You are not sure if an output anomaly is due to a new error or a side effect of an original error
  - Because inspection does not involve system execution, you do not need to worry about interactions between errors
- **Incomplete versions of a system can be inspected without additional costs**

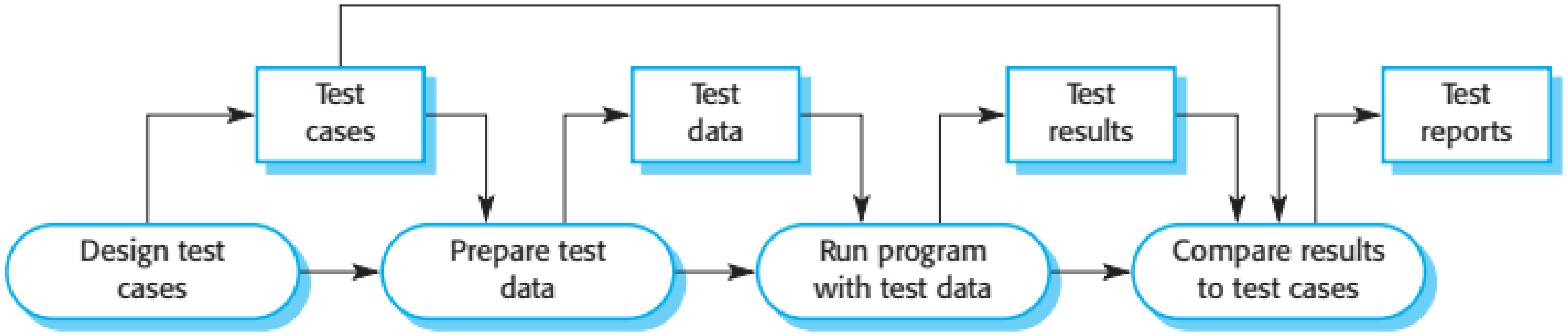
Advantages of software inspection (2)

- **An inspection can consider broader quality attributes of a program** such as
  - Compliance with standards
  - Portability
  - maintainability

Disadvantages of software inspection

- Inspections are not good for discovering defects that arise because of
  - unexpected interactions between different parts of a program
  - Timing problems
  - Problems with system performance
- In small development groups, it can be difficult and expensive to put together a separate inspection team because the members may also be developers of the software

A model of the Software Testing Process



Test planning

- Concerned with scheduling and resourcing all of the activities in a testing process
- Involves defining the testing process (takes into account the people and the time available)
- A test plan will be created that defines
  - What is to be tested
  - The predicted testing schedule
  - How tests will be recorded
  - Details of the tests to be run on the software (for critical systems)

Three stages of testing

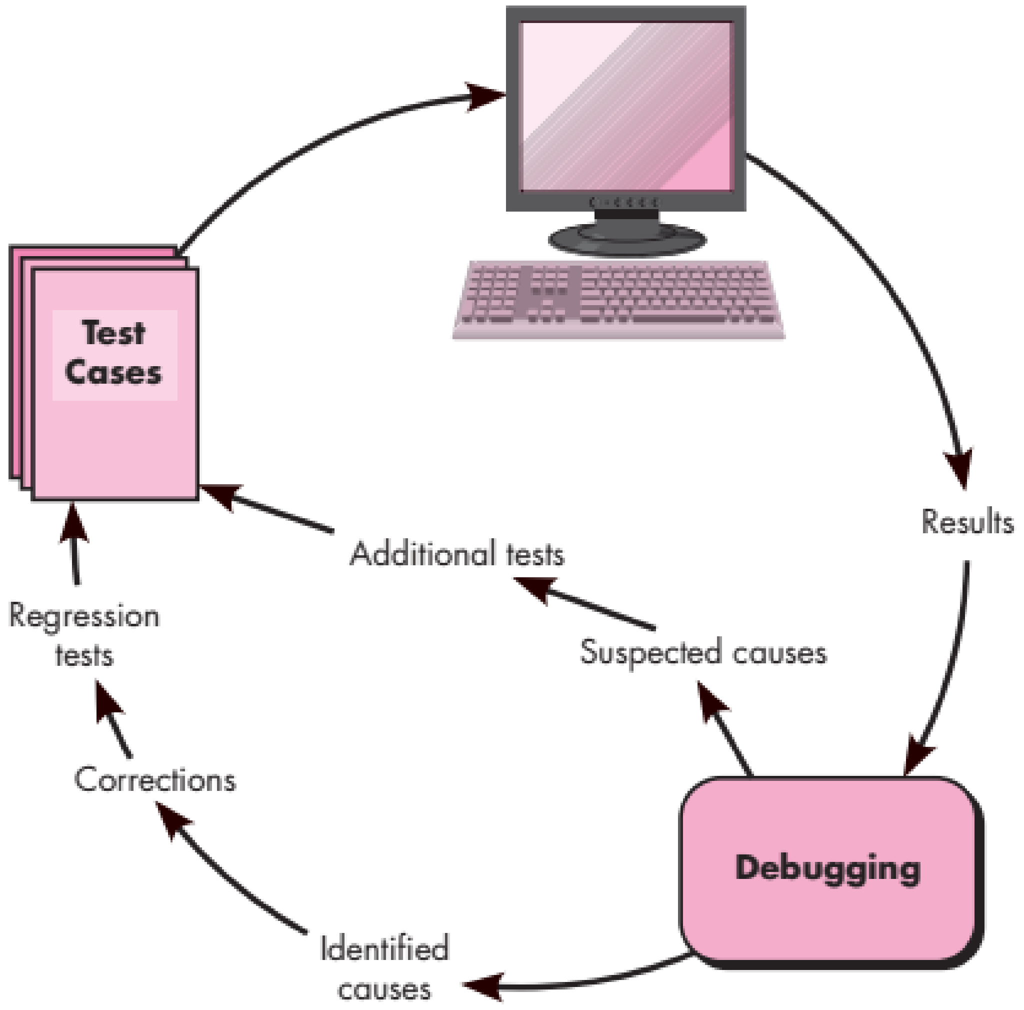
(typical for a commercial software system)

- DEVELOPMENT TESTING
- RELEASE TESTING
- USER TESTING

Debugging

- The process of fixing errors and problems that have been discovered by testing
- Debuggers
  - Use information from the program tests
  - Use their knowledge of the programming language and the intended outcome of the test to locate and repair the program error

The debugging process



R. Pressman

DEVELOPMENT TESTING

3 STAGES:

- UNIT TESTING
- COMPONENT TESTING
- SYSTEM TESTING

What it is…

- Includes all testing activities that are carried out by the development team
- The tester is usually the application developer/programmer who wrote the software
- Some development processes use programmer-tester pairs
  - Each programmer has an associated tester who develops tests and assists with the testing process
- For critical systems there is a separate testing group within the development team

Unit testing

- The process of **testing individual program units or components** (e.g., methods/functions or object classes)
  - Functions or methods are the simplest type of component
- **Tests for functions or methods** should be calls with different input parameters
- **Tests for object classes** should include
  - Testing all operations associated with the object
  - Setting and checking the value of all attributes associated with the object
  - Putting the object into all possible states (simulate all events that cause a state change)

Choosing effective unit test cases

- Test cases should show that, when used as expected, the component that you are testing does what it is supposed to do
  - these should reflect normal operation of a program and should show that the component works
- If there are defects in the component, these should be revealed by test cases
  - should be based on testing experience of where common problems arise
  - It should use abnormal inputs to check that these are properly processed and do not crash the component

Black box testing versus White box testing

Black box testing

- Involves testing a system with no prior knowledge of its internal workings
- A tester provides an input and observes the output generated by the system under test
- This makes it possible to identify how a system responds to expected and unexpected user actions, its response time, usability issues and reliability issues

White box testing

- Involves testing a system with detailed inside information of its source code, architecture and configuration
- It can expose issues like security vulnerabilities, broken paths or data flow issues

Many practitioners combine black box with white box testing

Black box testing pros and cons

https://www.imperva.com/learn/application-security/black-box-testing/#what-is-black-box-testing

Pros

- Testers do not require technical knowledge, programming or IT skills
- Testers do not need to learn implementation details of the system
- Tests can be executed by crowdsourced or outsourced testers
- Low chance of false positives
- Tests have lower complexity, since they simply model common user behavior
- Tests may be conducted at low scale or on a non-production-like environment

Cons

- Difficult to automate
- Requires prioritization, typically infeasible to test all user paths
- Difficult to calculate test coverage
- If a test fails, it can be difficult to understand the root cause of the issue

Gray (grey) box testing (1)

- A compromise between black box and white box testing
- Tests applications and environments with partial knowledge of internal workings of the system
- Commonly used for
  - Penetration testing (a.k.a, pen test)
    - a simulated cyber attack against your computer system to check for exploitable vulnerabilities
  - End-to-end system testing
  - Integration testing

Gray (grey) box testing (2)

- Most effective for evaluating
  - Web applications
  - Integration testing
  - Distributed environments
  - Business domain testing
  - Performing security assessments

Component testing

- **A process where several individual units are integrated to create composite components**
- Should **focus on testing the component interfaces** that provide access to the component functions

Types of interface between program components (1)

- **Parameter interfaces**
  - Interfaces in which data or sometimes function references are passed from one component to another
  - methods in an object have a parameter interface
- **Shared memory interfaces**
  - interfaces in which a block of memory is shared between components
  - data is placed in the memory by one subsystem and retrieved from there by other subsystems
  - used in embedded systems, where sensors create data that is retrieved and processed by other system components

Types of interface between program components (2)

- **Procedural interfaces**
  - interfaces in which one component encapsulates a set of procedures that can be called by other components
  - objects and reusable components have this form of interface
- **Message passing interfaces**
  - interfaces in which one component requests a service from another component by passing a message to it
    - A return message includes the results of executing the service
  - some object-oriented and client-server systems have this form of interface

Classes of interface errors (1)

- **Interface misuse**
  - a calling component calls some other component and makes an error in the use of its interface
  - **common in parameter interfaces**
    - Parameters may be of the wrong type
    - Parameters may be passed in the wrong order
    - The wrong number of parameters may be passed

Classes of interface errors (2)

- **Interface misunderstanding**
  - A calling component misunderstands the specification of the interface of the called component and makes assumptions about its behavior
  - The called component does not behave as expected, which then causes unexpected behavior in the calling component
  - *Example:* A failure in searching when a binary search method may be called with a parameter that is an unordered array

Classes of interface errors (3)

- **Timing errors**
  - These occur in real-time systems that use a shared memory or a message-passing interface
  - The producer of data and the consumer of data may operate at different speeds
  - Unless particular care is taken in the interface design, the consumer can access out-of-date information because the producer of the information has not updated the shared interface information

Guidelines for interface testing (1)

- **Examine the code to be tested and identify each call to an external component**
  - Design a set of tests in which the values of the parameters to the external components are at the **extreme ends of their ranges**
  - Extreme values will most likely reveal interface inconsistencies
- **Where pointers are passed across an interface, always test the interface with null pointer parameters**

Guidelines for interface testing (2)

- **Where a component is called through a procedural interface, design tests that deliberately cause the component to fail**
  - **Differing failure assumptions** are one of the most common specification misunderstandings
- **Use stress testing in message passing systems**
  - This means that you should design tests that generate many more messages than are likely to occur in practice
  - This is an effective way of revealing timing problems

Guidelines for interface testing (3)

- **Where several components interact through shared memory, design tests that vary the order in which these components are activated**
  - These tests may reveal implicit assumptions made by the programmer about the order in which the shared data is produced and consumed

Sometimes it is **better to use inspections and reviews rather than testing to look for interface errors.**

Inspections can concentrate on component interfaces and questions about the assumed interface behavior asked during the inspection process.

System testing

- **where some or all of the components in a system are integrated and the system is tested as a whole**
- should **focus on testing component interactions**
- checks that components are
  - compatible
  - interact correctly, and
  - transfer the right data at the right time across their interfaces

Difference of system testing from component testing

- reusable components (that have been separately developed) and off-the-shelf systems may be integrated with newly developed components
- Components developed by different team members or sub-teams may be integrated at this stage
  - System testing is a collective rather than an individual process
  - System testing may involve a separate testing team with no involvement from designers and programmers

Emergent behavior of systems (1)

- This means that some system functionality and characteristics only become obvious when you put the components together
- **Planned emergent behavior**
  - Has to be tested
  - *Example*: integrating an authentication component with a component that updates the system database
    - system feature that restricts information updating to authorized users

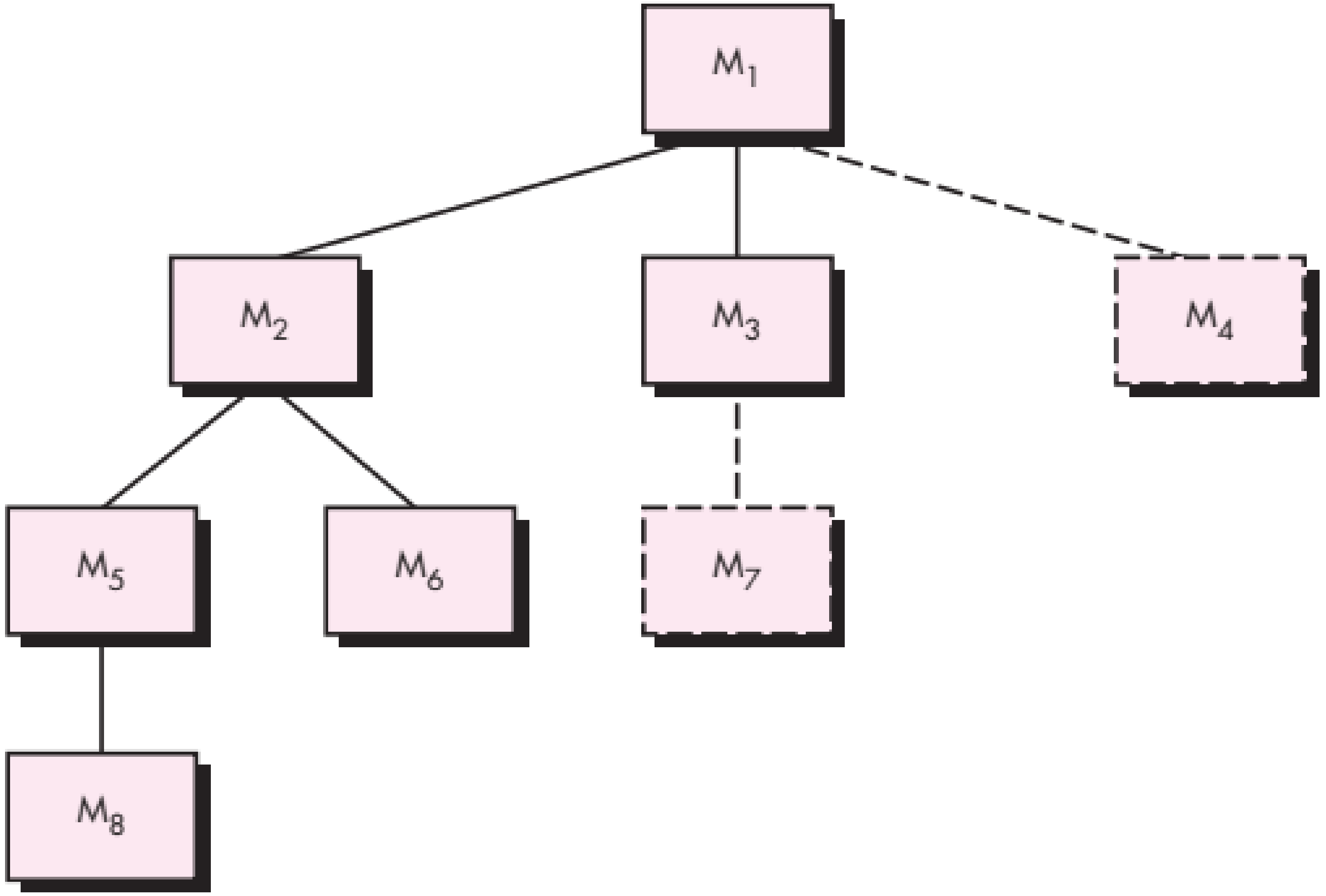
Emergent behavior of systems (2)

- **Unplanned and unwanted emergent behavior**
  - Tests need to be developed to check that the system is only doing what it is supposed to do

Incremental integration and testing

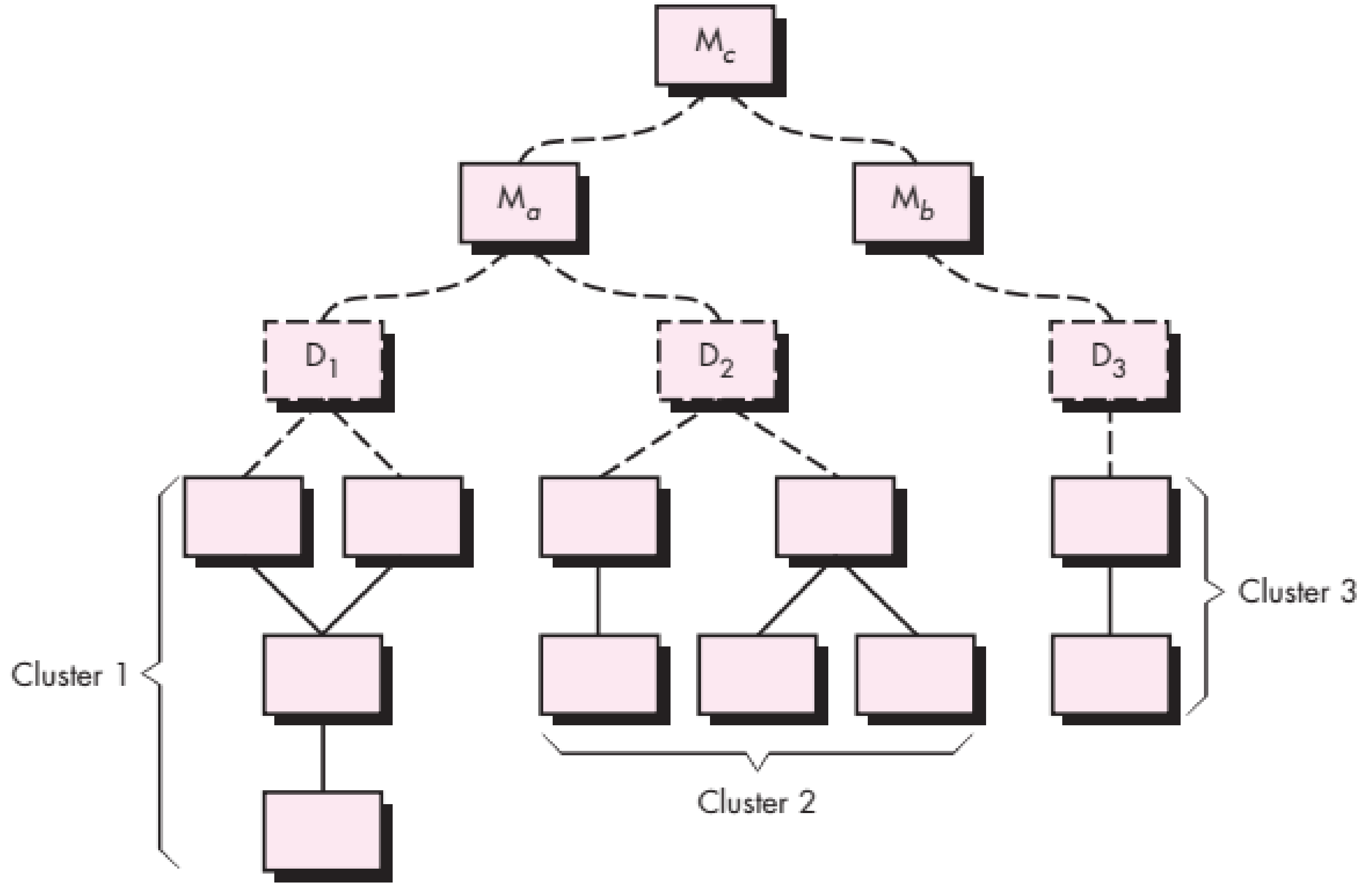
- This approach should always be used during system testing
- *How it is done*: Integrate a component, test the system, integrate another component, test again, and so on…
- **If problems occur, they are probably due to interactions with the most recently integrated component**
- This kind of testing is **fundamental to agile methods**, where regression tests are run every time a new increment is integrated

Top-down Integration testing



R. Pressman

Bottom-up Integration Testing



R. Pressman

RELEASE TESTING

- REQUIREMENTS-BASED TESTING
- SCENARIO TESTING
- PERFORMANCE TESTING

What it is… (1)

- the process of testing a particular release of a system that is intended for use outside of the development team
- Normally, the system release is for customers and users
- In a complex project, the release could be for other teams that are developing related systems
- For software products, the release could be for product management who then prepare it for sale

What it is… (2)

- Release testing is usually a black-box testing process where tests are derived from the system specification
  - The system is treated as a black box whose behavior can only be determined by studying its inputs and the related outputs
- Another name: **functional testing**
  - so-called because the tester is only concerned with functionality and not the implementation of the software

What it is… (3)

- **The primary goal**: to convince the supplier of the system that it is good enough for use so it can be released as a product or delivered to the customer
- Release testing has to show that
  - the system delivers its specified functionality, performance, and dependability, and
  - it does not fail during normal use

Difference from system testing

- The system development team should not be responsible for release testing
- Release testing is a *process of validation checking to ensure that a system meets its requirements and is good enough for use by system customers*
  - System testing by the development team should *focus on discovering bugs in the system* (**defect testing**)

Requirements-based testing (1)

- A general principle of good requirements engineering practice is that **requirements should be testable**
  - the requirement should be written so that a test can be designed for that requirement
  - A tester can then check that the requirement has been satisfied

Requirements-based testing (2)

- a systematic approach to test-case design **where you consider each requirement and derive a set of tests for it**
- A validation testing rather than defect testing
- You have to
  - demonstrate that the system has properly implemented its requirements
  - write several tests to ensure that you have coverage of the requirement
  - keep traceability records of your requirements-based testing, which link the tests to the specific requirements that you have tested

Scenario testing (1)

- An approach to release testing whereby you devise typical scenarios of use and use these scenarios to develop test cases for the system
- A **scenario** is a story that describes one way in which the system might be used
- Scenarios should be **realistic**, and real system users should be able to relate to them

Scenario testing (2)

- If scenarios or user stories have been used as part of the requirements engineering process, they may be reused as testing scenarios
- When a scenario-based approach is used, several requirements are normally tested within the same scenario
- While checking individual requirements, combinations of requirements are also tested to see to it that they do not cause problems

Performance testing (1)

- Performance tests have to be designed to ensure that the system can process its intended load
- This usually involves running a series of tests where the load is increased until the system performance becomes unacceptable
- Concerned both with demonstrating that the system meets its requirements and discovering problems and defects in the system

Performance testing (2)

- To test whether performance requirements are being achieved, an operational profile may need to be constructed
- **Operational profile** – a set of tests that reflect the actual mix of work that will be handled by the system

Stress testing (1)

- testing the system by making demands that are outside the design limits of the software
- Stress testing helps you do two things:
  - **Test the failure behavior of the system**
    - Example: the system exceeds the maximum anticipated load
      - system failure should not cause data corruption or unexpected loss of user services
      - Stress testing checks that overloading the system causes it to "fail-soft" rather than collapse under its load

Stress testing (2)

- **Reveal defects that only show up when the system is fully loaded**
  - Although these defects are unlikely to cause system failures in normal use, there may be unusual combinations of circumstances that the stress testing replicates
- Stress testing is particularly relevant to distributed systems based on a network of processors
  - distributed systems often exhibit severe degradation when they are heavily loaded
  - the network becomes swamped with coordination data that the different processes must exchange

*** From Roger Pressman *** (1)

- System testing
  - a series of different tests whose primary purpose is to fully exercise the computer-based system
  - Tests under this testing process:
    - **Recovery Testing**
      - a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic (performed by the system itself), reinitialization, checkpointing mechanisms, data recovery, and restart are evaluated for correctness

*** From Roger Pressman *** (2)

- **Security Testing**
  - attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration
  - the tester plays the role(s) of the individual who desires to penetrate the system
- **Stress Testing**
  - executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
  - Variation: ***sensitivity testing*** (attempts to uncover data combinations within valid input classes that may cause instability or improper processing)

*** From Roger Pressman *** (3)

- **Performance Testing**
  - occurs throughout all steps in the testing process
  - designed to test the run-time performance of software within the context of an integrated system
  - often coupled with stress testing and usually require both hardware and software instrumentation
- **Deployment Testing**
  - sometimes called configuration testing
  - exercises the software in each environment in which it is to operate
  - examines all installation procedures and specialized installation software (e.g., "installers") that will be used by customers, and all documentation that will be used to introduce the software to end users

USER TESTING

- ALPHA TESTING
- BETA TESTING
- ACCEPTANCE TESTING

What it is…

- a stage in the testing process in which users or customers provide input and advice on system testing
- Can be formal or informal
- Important because influences from the user's working environment can have a major effect on the reliability, performance, usability, and robustness of a system
  - It is practically impossible for a system developer to replicate the system's working environment, as tests in the developer's environment are inevitably artificial

Alpha testing

- where a selected group of software users work closely with the development team to test early releases of the software
- Users and developers work together to test a system as it is being developed
  - the users can identify problems and issues that are not readily apparent to the development testing team
  - developers can only really work from the requirements, but these often do not reflect other factors that affect the practical use of the software
  - users cam provide information about practice that helps with the design of more realistic tests

Beta testing (1)

- where a release of the software is made available to a larger group of users to allow them to experiment and to raise problems that they discover with the system developers
- takes place when an early, sometimes unfinished, release of a software system is made available to a larger group of customers and users for evaluation
  - Beta testers may be a selected group of customers who are early adopters of the system
  - the software may also be made publicly available for use by anyone who is interested in experimenting with it

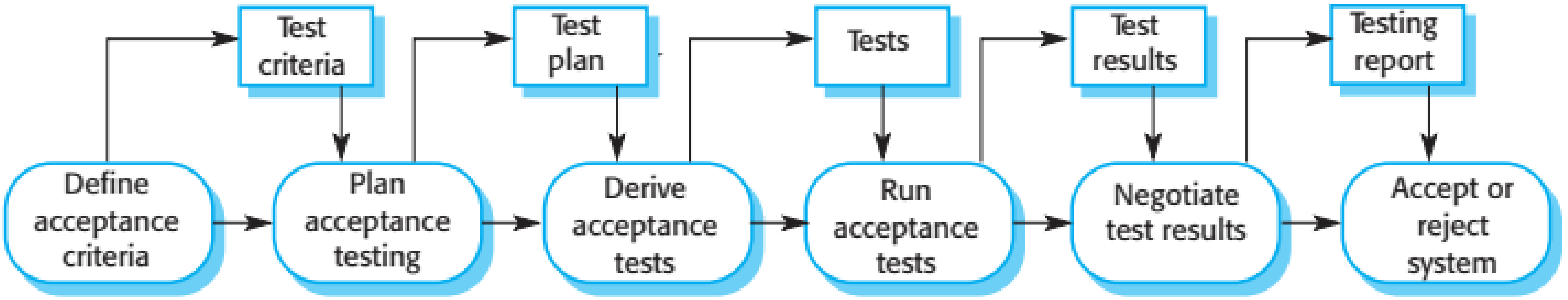
Beta testing (2)

- mostly used for software products that are used in many different settings
- Also used to discover interaction problems between the software and features of its operational environment
- Also a form of marketing
  - Customers learn about their system and what it can do for them

Acceptance testing

- where customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment
- an inherent part of custom systems development where customers test a system, using their own data, and decide if it should be accepted from the system developer
- **Acceptance implies that final payment should be made for the software**

The acceptance testing process (1)



The acceptance testing process (2)

- **Define acceptance criteria**
  - This stage should ideally take place early in the process before the contract for the system is signed
  - The acceptance criteria should be part of the system contract and be approved by the customer and the developer

The acceptance testing process (3)

- **Plan acceptance testing**
  - This stage involves deciding on the resources, time, and budget for acceptance testing and establishing a testing schedule
  - The acceptance test plan should also discuss the required coverage of the requirements and the order in which system features are tested
  - It should define risks to the testing process such as system crashes and inadequate performance, and discuss how these risks can be mitigated

The acceptance testing process (4)

- **Derive acceptance tests**
  - Once acceptance criteria have been established, tests have to be designed to check whether or not a system is acceptable
  - Acceptance tests should aim to test both the functional and non-functional characteristics (e.g., performance) of the system
  - They should ideally provide complete coverage of the system requirements

The acceptance testing process (5)

- **Run acceptance tests**
  - The agreed acceptance tests are executed on the system
  - Ideally, this step should take place in the actual environment where the system will be used, but this may be disruptive and impractical
  - A user testing environment may have to be set up to run these tests
  - Difficult to automate this process because it may involve testing the interactions between end-users and the system
  - Some training of end-users may be required

The acceptance testing process (6)

- **Negotiate test results**
  - It is very unlikely that all of the defined acceptance tests will pass and that there will be no problems with the system
    - If this is the case, then acceptance testing is complete and the system can be handed over
  - More commonly, some problems will be discovered
    - In such cases, the developer and the customer have to negotiate to decide if the system is good enough to be used
    - They must also agree on how the developer will fix the identified problems.

The acceptance testing process (7)

- **Reject/accept system**
  - This stage involves a meeting between the developers and the customer to decide on whether or not the system should be accepted
  - If the system is not good enough for use, then further development is required to fix the identified problems
    - Once complete, the acceptance testing phase is repeated

Test Strategies for WebApps

Steps (1)

- The content model for the WebApp is reviewed to uncover errors
- The interface model is reviewed to ensure that all use cases can be accommodated.
- The design model for the WebApp is reviewed to uncover navigation errors.
- The user interface is tested to uncover errors in presentation and/or navigation mechanics.

Steps (2)

- Each functional component is unit tested.
- Navigation throughout the architecture is tested.
- The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration.
- Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment.

Steps (3)

- Performance tests are conducted.
- The WebApp is tested by a controlled and monitored population of end users
  - The results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns, and WebApp reliability and performance.

Because many WebApps evolve continuously, the testing process is an **ongoing activity**, conducted by support staff who use regression tests derived from the tests developed when the WebApp was first engineered.

Strategic Issues in Software Testing

A software testing strategy will succeed when software testers: (1) (Tom Gilb)

- Specify product requirements in a quantifiable manner long before testing commences
- State testing objectives explicitly
- Understand the users of the software and develop a profile for each user category
- Develop a testing plan that emphasizes "rapid cycle testing"
- Build "robust" software that is designed to test itself

A software testing strategy will succeed when software testers: (2) (Tom Gilb)

- Use effective technical reviews as a filter prior to testing
- Conduct technical reviews to assess the test strategy and test cases themselves
- Develop a continuous improvement approach for the testing process

End of Presentation