

Software Process Modeling

Software Testing and Maintenance

Session Outcomes

- Testing
 - V & V
 - Types of testing
 - Black Box testing
 - White box testing
 - Software Testing Strategies
 - Unit testing
 - Integration Testing
 - System Testing
 - Acceptance Testing
- Maintenance

Story So Far ...

- You have already learned how to carry out
 - Feasibility Study
 - Requirements Gathering, Analysis and Specification
 - Design
 - Implementation
- Now you have an implemented system with you to start testing.....

Importance of testing

- Defective Software

- In 1992, Mary Bandar received an invitation to attend a kindergarten in Winona, Minnesota, along with others born in '88. Mary was born in 1888 and 104 years old at the time.

- <http://www.apnewsarchive.com/1993/Woman-Born-in-1888-Gets-Surprise-Invitation-To-Kindergarten/id-a0d5e1ddb4be59cdc6b066d0e9517e29>

- An F-18 crashed because of a missing exception condition: if ... then ... without the else clause that was thought could not possibly arise.

- **Computer-Related Risks** -By Peter G. Neumann

What is Software Testing?

- “Software Testing is the process of executing a program or system with the **intent of finding errors**” [Myers, 79].
- “Program testing can be a very effective way to show the presence of bugs, but it is hopelessly **inadequate for showing their absence**” [Dijkstra, 1972]

Verification vs validation

- Verification:

"Are we building the product right"

- The software should conform to its specification – functional and non-functional requirements

- Validation:

"Are we building the right product"

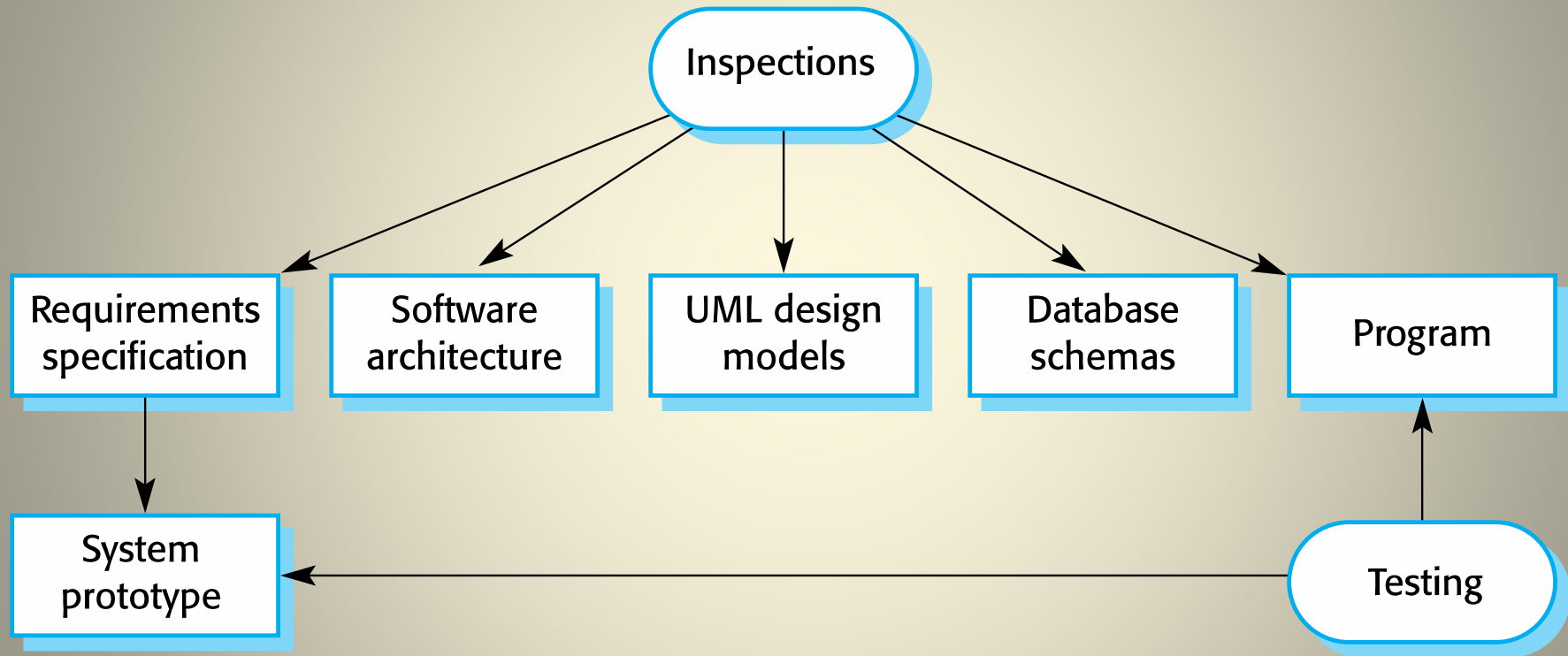
- The software should do what the user really requires which **might** be different from specification

V & V Techniques

- *Software inspections* - Concerned with analysis of the static system representation to discover problems (static verification)
 - No need to execute a software to verify it.
- *Software testing* - Concerned with exercising and observing product behaviour (dynamic verification)
 - The system is executed with test data and its operational behaviour is observed

Ref: Software Engineering, I. Sommerville, 10th Edition

Inspections and testing



Ref: Software Engineering, I. Sommerville, 10th Edition

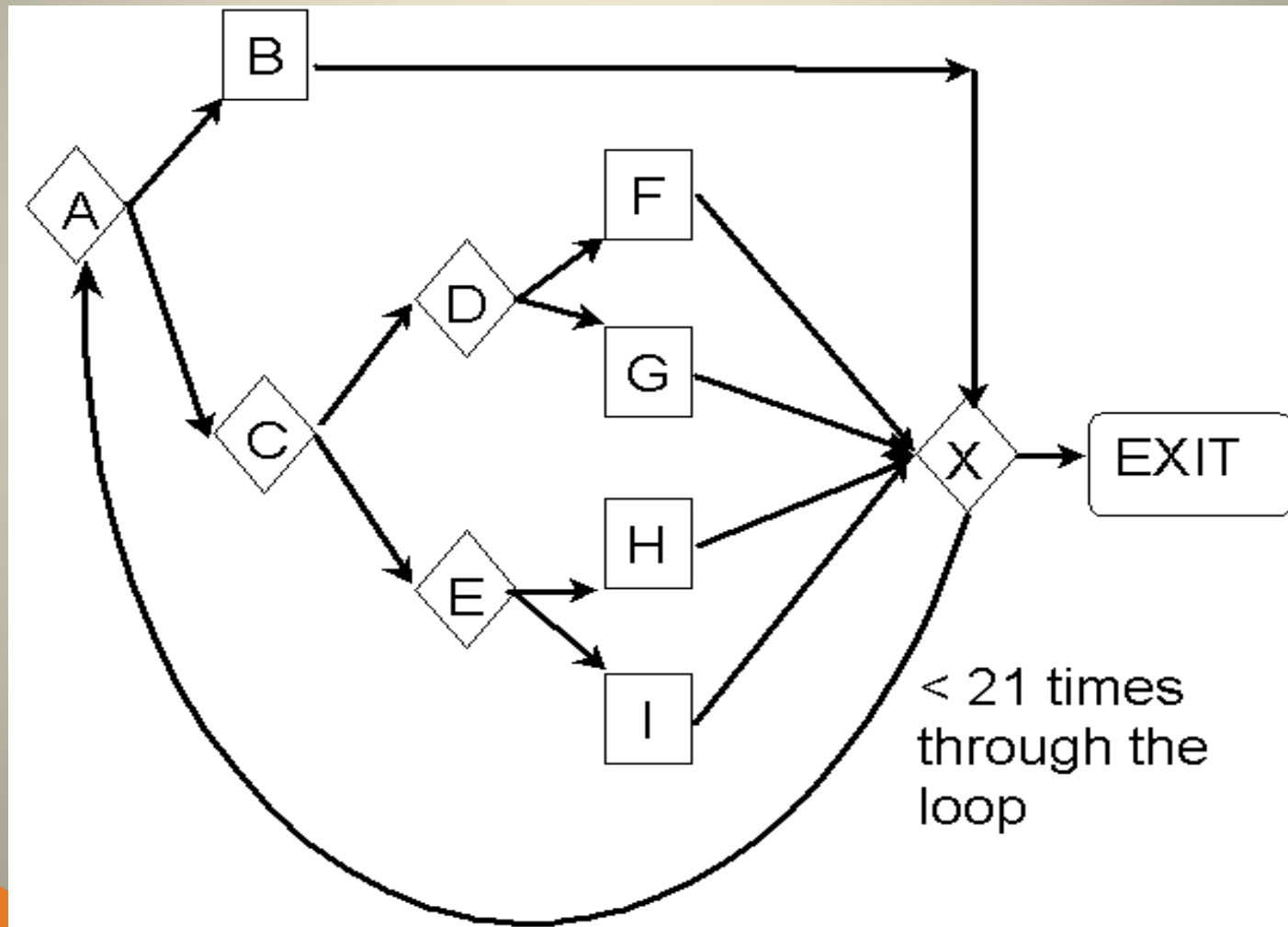
Think...

- Can we have a 100% error free s/w?
 - It is not possible to *prove* that there are no faults in the software using testing
- Why?

Why Can't Every Defect be Found?

- Too many possible paths.
- Too many possible inputs.
- Too many possible user environments.

Too Many Possible Paths

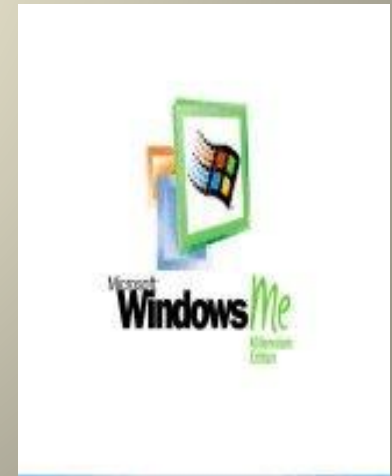
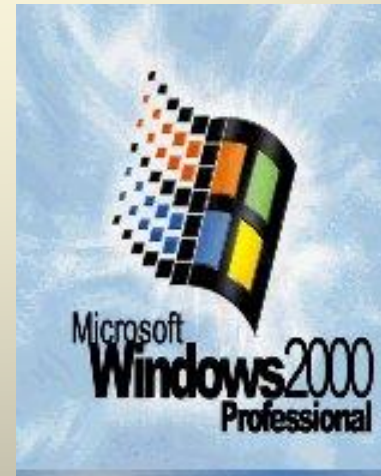
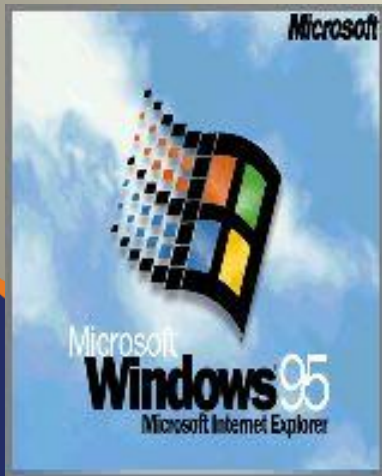


Too Many Possible Inputs

- Programs take input in a variety of ways: mouse, keyboard, and other devices.
- Must test Valid and Invalid inputs.
- Most importantly, there are an infinite amount of sequences of inputs to be tested.

Too Many Possible User Environments

- Difficult to replicate the user's combination of hardware, peripherals, OS, and applications.
- Impossible to replicate a thousand-node network to test networking software.



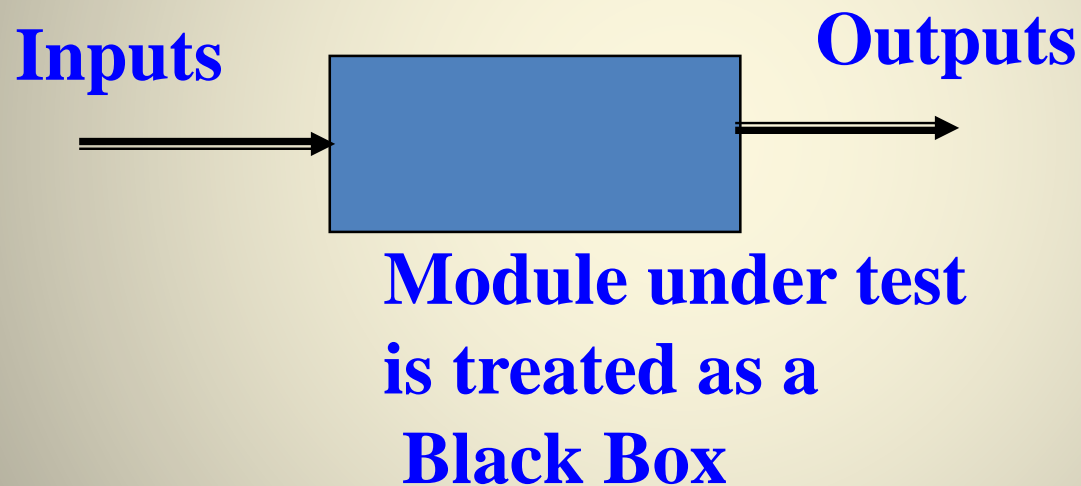
Types of Testing

- Exhaustively testing of all possible input/output combinations is too expensive
- There are 2 types of testing.
 - Black box testing (Functional testing)
 - Structural testing (White box testing)

Ref: Software Engineering, I. Sommerville, 10th Edition

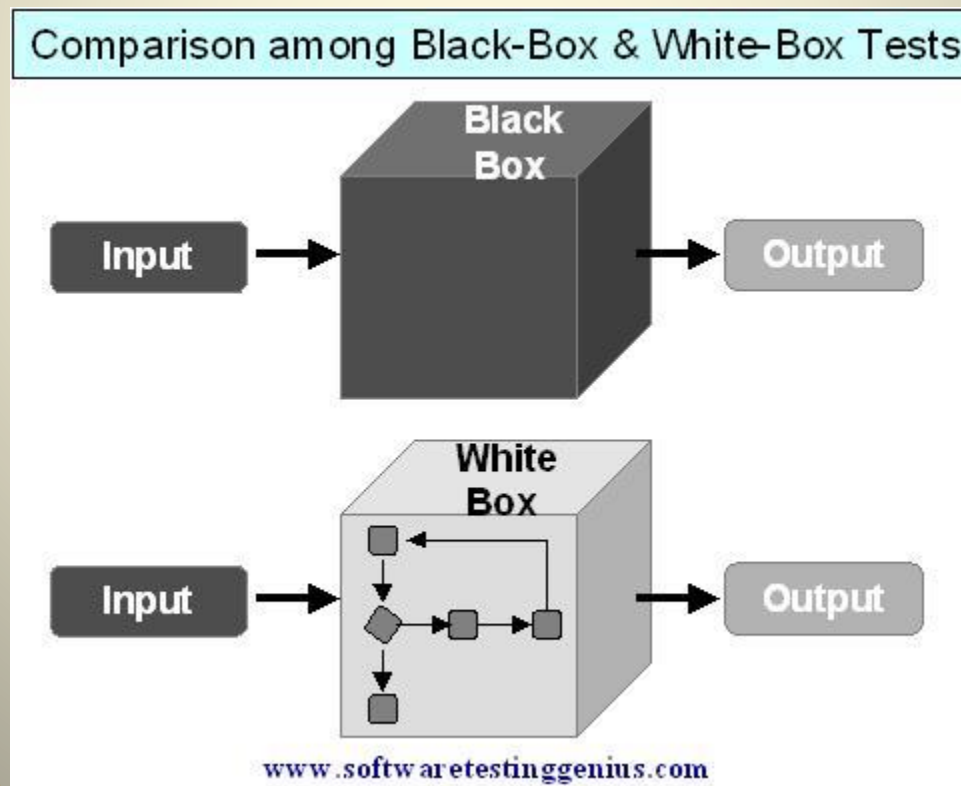
Black Box Testing

- Testing focus on the software functional requirements, and input/output.



White box testing

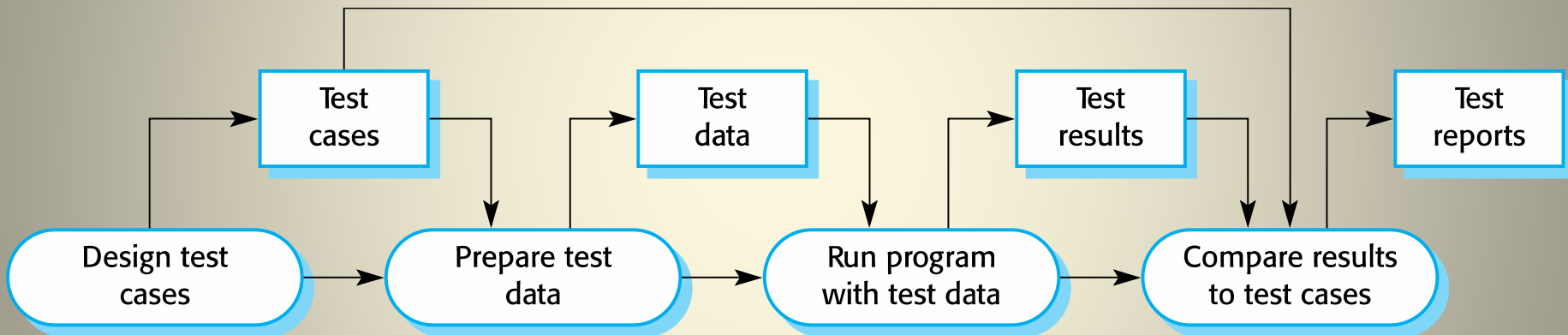
- Testing is based on the structure of the program.
 - In white box testing internal structure of the program is taken into account
 - The test data is derived from the structure of the software



Activity 1

- What are the differences between Black Box and White box Testing ?

A model of the software testing process



Ref: Software Engineering, I. Sommerville, 10th Edition

Black box testing strategies

- **Equivalence Class Testing:**
 - It is used to minimize the number of possible test cases to a best level while maintaining reasonable test coverage.
- **Boundary Value Testing:**
 - Boundary value testing is focused on the values at boundaries.
 - This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases.

Equivalence partitioning

Example

- Example of a function which takes a parameter “month”.
- The valid range for the month is 1 to 12, representing January to December. This valid range is called a partition.
- In this example there are two further partitions of invalid ranges.

$$x < 1$$

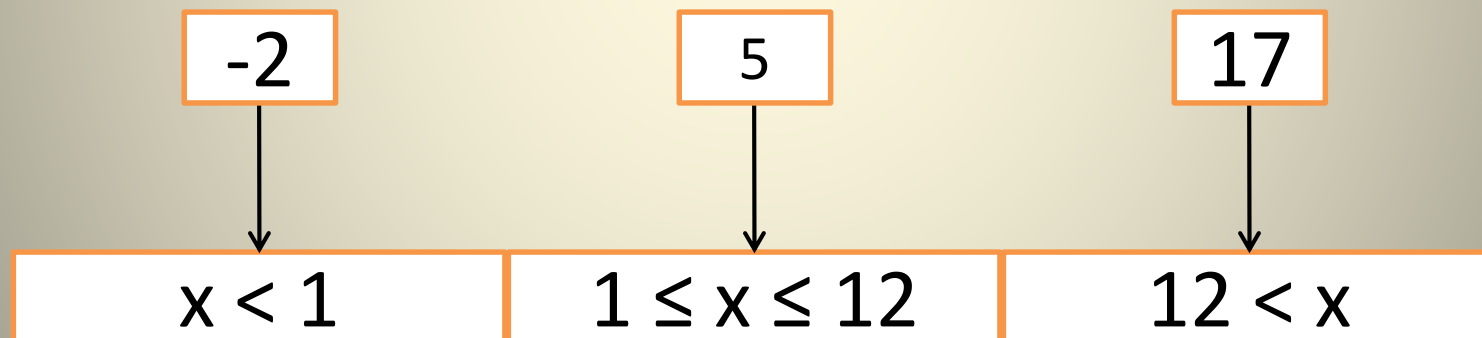
$$1 \leq x \leq 12$$

$$12 < x$$

Equivalence partitioning

Example

- Test cases are chosen so that each partition would be tested.



Equivalence partitions

- In equivalence-partitioning technique we need to test only one condition from each partition.
 - This is because we are assuming that all the conditions in one partition will be treated in the same way by the software.
- Equivalence partitioning is an effective approach to testing because it helps to reduce the number of tests.

Equivalence partitioning

Example

- In company XYZ, to become a staff member one has to be 18 years old. The retirement age is 57.
 - Find the test values for the employment age.

Activity

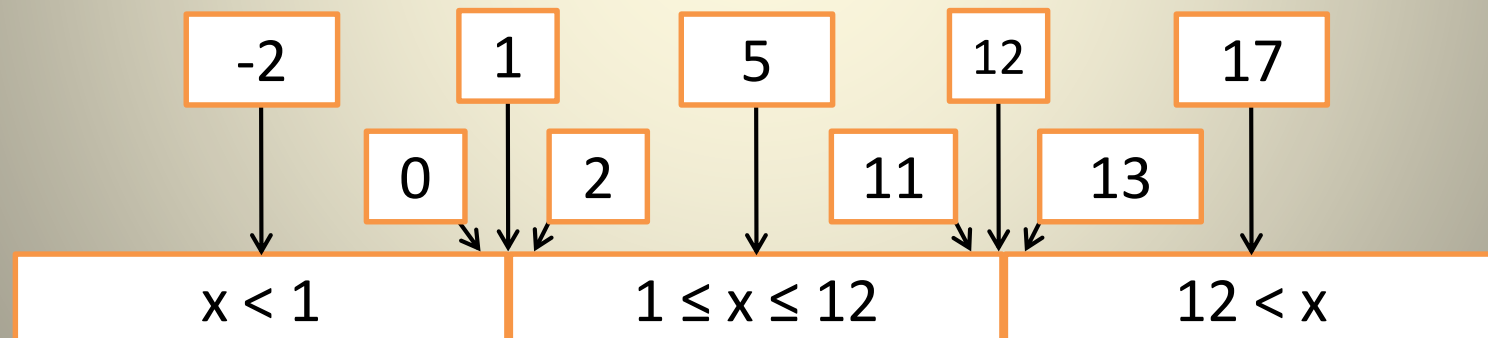
- In a Library, students can borrow books. There is a limit given for students on the number of books they can borrow at one time. A student can borrow 2-5 books at one time.
- With equivalence partitioning identify the test values to check the borrowing **limit** of a student at **one time**.

Boundary value analysis

- Equivalence partitioning is not a stand alone method to determine test cases. It is usually supplemented by ***boundary value analysis***.
- ***Boundary value analysis*** focuses on values on the edge of an equivalence partition or at the smallest value on either side of an edge.

Equivalence partitioning with boundary value analysis

- Example of a function which takes a parameter “month”.
- Test cases are supplemented with ***boundary values***.



Boundary Value Analysis Example

- Write test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis

Boundary Value Analysis - Example

- Test Cases
 - 1: Consider test data exactly as the **input boundaries** of input domain i.e. values 1 and 1000.
 - 2: Consider test data with values just **below the extreme edges** of input domains i.e. values 0 and 999.
 - 3: Consider test data with values just **above the extreme edges** of input domain i.e. values 2 and 1001.

Activity

An online Movie ticket booking system lets users book movie tickets for currently playing movies and newly releasing movies. When a user is going to make a booking, he/she must select type of movie (English/Tamil/Hindi) and enter Type of ticket (Full/Half). Also must enter No of Tickets wanted (1 to 10) and if taking any children (age 3 to 12) must fill as “Yes” if not “No”. The maximum no of tickets allowed for a booking is 10. The system interface shows multiple text boxes for the user to enter the needed values.

Write 8 Sample test data to check the format and correctness of user-entered data to the above system using equivalence partitioning and Boundary Value Analysis. Use the Format given below.

Software Testing Levels

- Unit testing : Individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
- Integration testing : Several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
- System testing: Some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.
- Acceptance Testing: 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. Primarily for custom systems.

Unit testing

- Unit testing is the process of testing individual components in isolation.
- Unit testing can be automated, but may still be performed manually.
- Units may be:
 - Individual functions or methods within an object
 - Object classes with several attributes and methods

Ref: Software Engineering, I. Sommerville, 10th Edition

Assert

- Last semester in IP module you used “assert” expression.
 - It can be used as a means of checking the programs expected behavior.
 - It can be used for unit testing.

`assert(<boolean expression>)`

Introduction

- This simple function requires a boolean (conditional) expression.
- An assertion is a fact that we strongly believe to be true.
 - e.g. `assert(5 > 2)` or `assert(20 == 20)`
- When your program is executed the assertion is checked and if the condition is true, it simply moves to the next statement in the code.
- The `assert()` function requires the `<assert.h>` header file.
- You can disable assertions from production code by using the following directive before `#include <assert.h>`
`#define NDEBUG`

Quiz

- What will be the outcome of the following assertions (pass/fail).
- Assume that `char grade = 'B'; int radius= 254; int perimeter = 40`
 1. `assert(grade == 'A') ;`
 2. `assert(grade == 'A' || grade == 'B' || grade = 'C') ;`
 3. `assert(radius == 34) ;`
 4. `assert(radius == 254) ;`
 5. `assert(perimeter < 50) ;`

Activity

- Write a unit testing function for the following program using “assert”.

```
interface Adder {  
    int add(int a, int b);  
}  
class AdderImpl implements Adder {  
    int add(int a, int b) {  
        return a + b;  
    }  
}
```

Unit Testing

Method Used for unit testing: White Box Testing method is used for executing the unit test.

When should Unit testing be done?

Unit testing should be done before Integration testing.

By whom should unit testing be done?

Unit testing should be done by the developers.

Advantages of Unit testing:

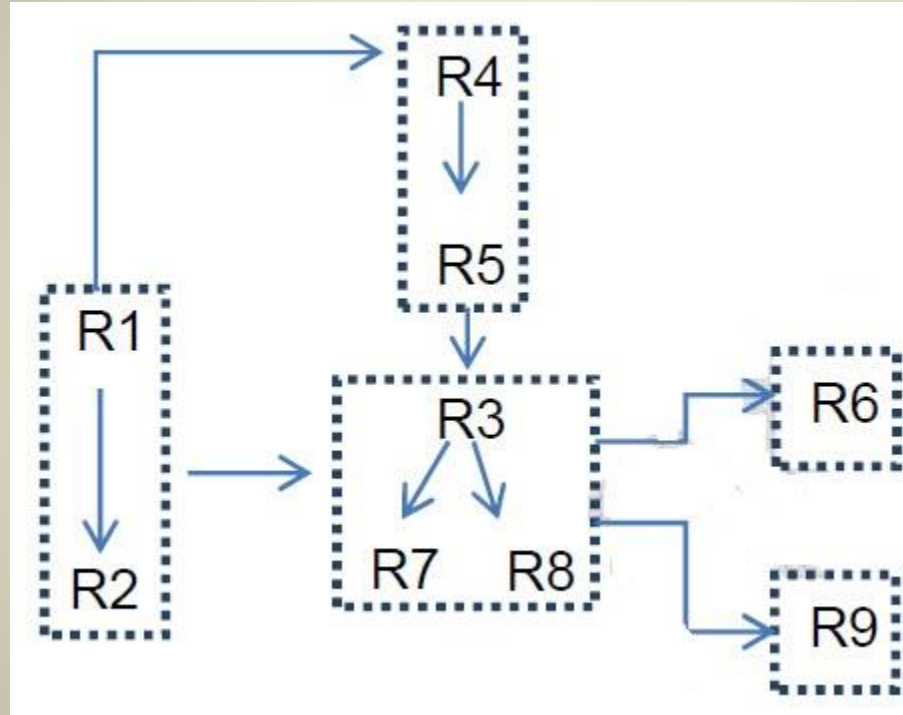
1. Issues are found at early stage. Since unit testing are carried out by developers where they test their individual code before the integration. Hence the issues can be found very early and can be resolved then and there without impacting the other piece of codes.
2. Unit testing helps in maintaining and changing the code.
3. Since the bugs are found early in unit testing it also helps in reducing the cost of bug fixes.

Integration Testing

- Integration testing is the phase in software testing in which individual software modules are combined and tested as a group.
- It occurs **after** unit testing and **before** system testing.
- Integration Testing
 - Big Bang Testing
 - Top Down Testing
 - Bottom Up Testing

Activity

- Requirements logical dependencies (A \rightarrow B means that B is dependent on A.)



How would you structure the test execution schedule according to the above criteria?

Big Bang Testing

- In Big Bang Integration testing, individual modules of the programs are not integrated until **every thing** is ready. It is called 'Run it and see' approach.
- In this approach, the program is integrated **without any formal integration testing**, and then run to ensures that all the components are working properly.



Activity

- **Big Bang Approach** : In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole.
- Is it a good method? Why ?

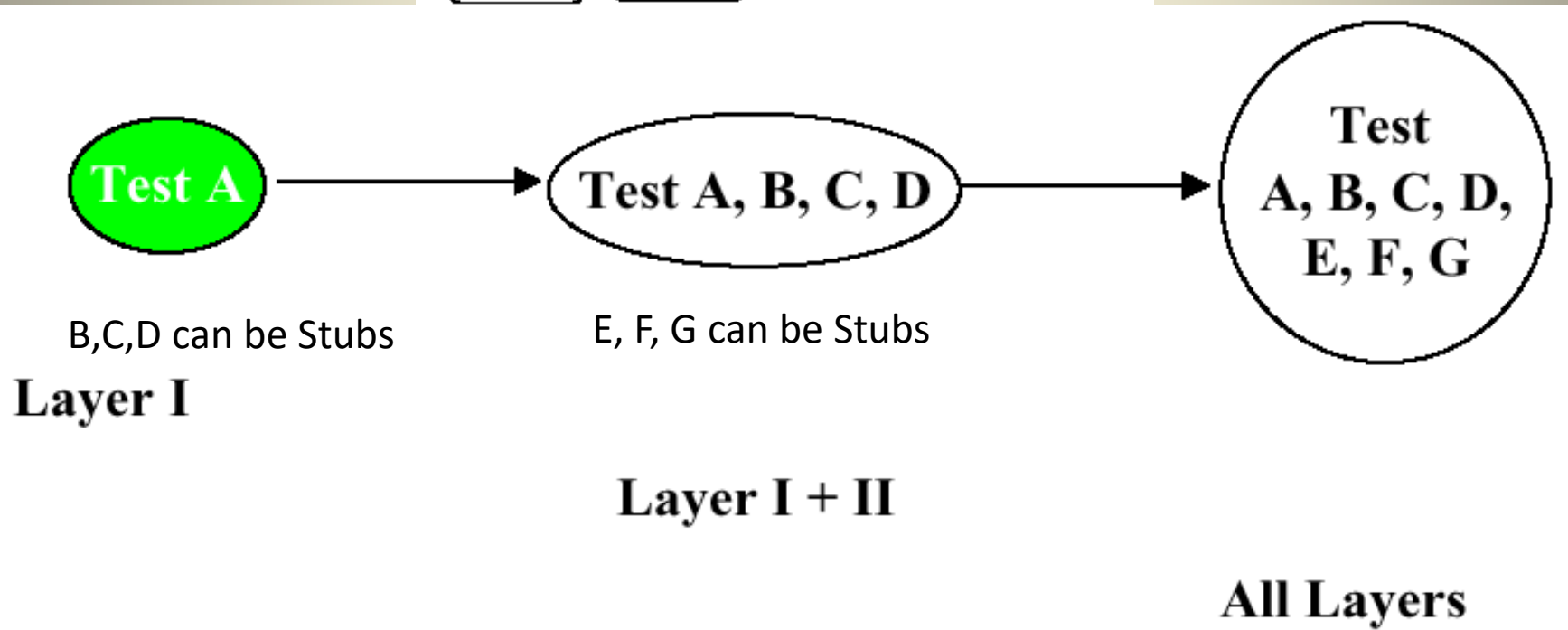
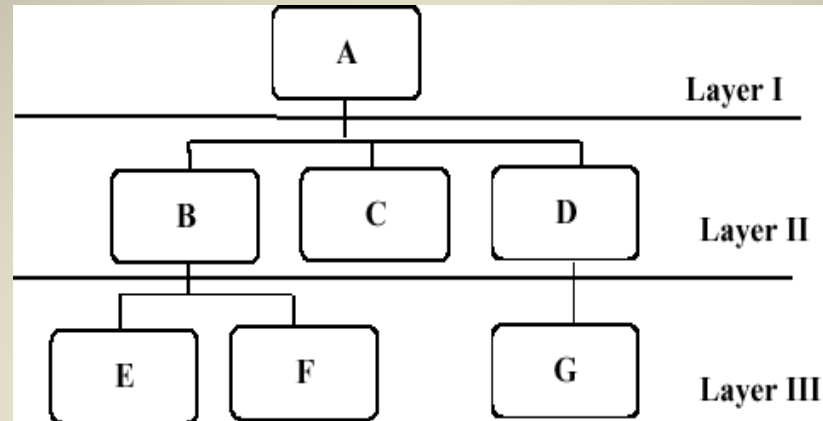
Activity Answer

- **Disadvantage:** The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.
- The chances of having critical failures are more because of integrating all the components together at same time.

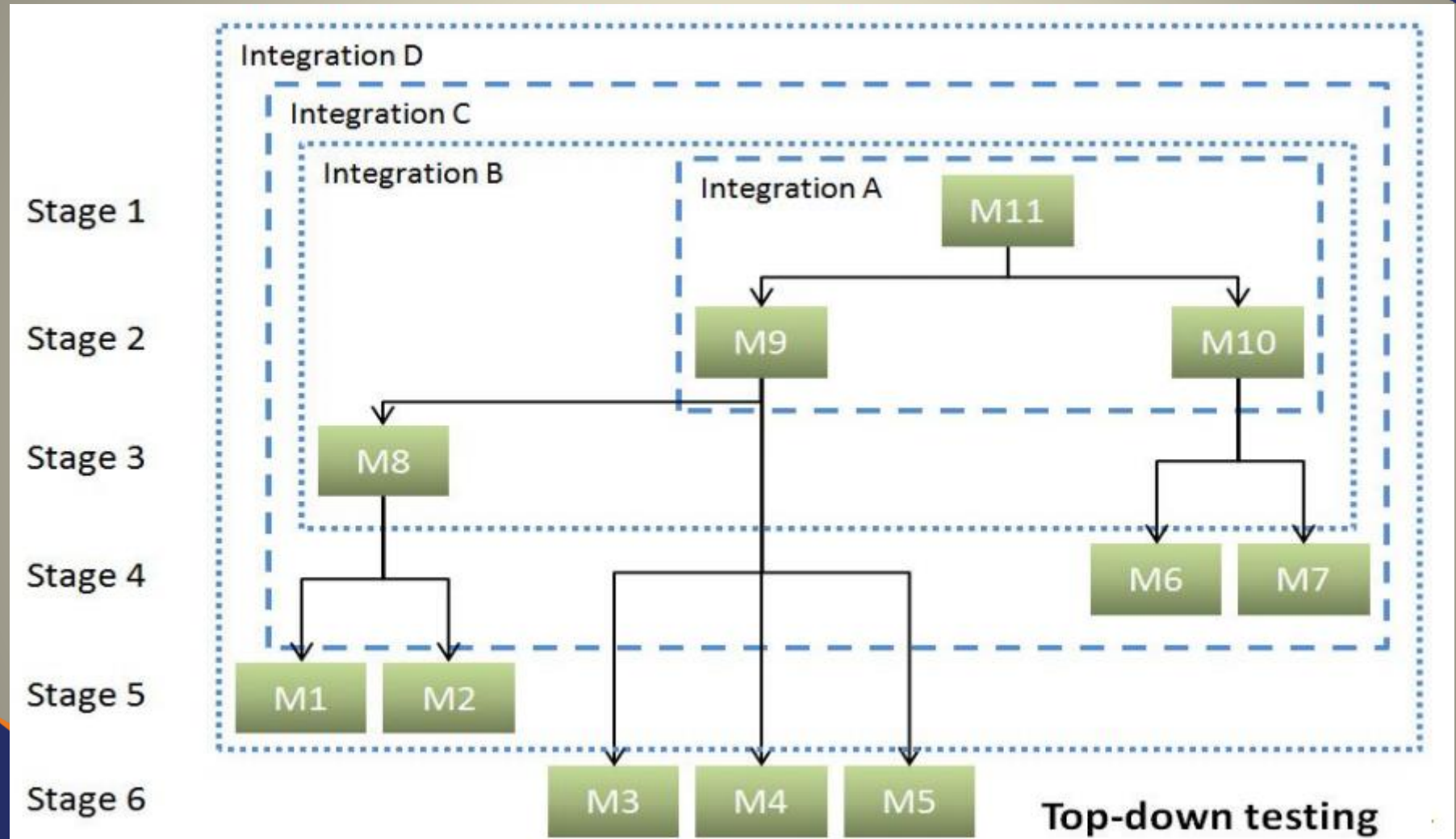
Top-down Testing

- Top down integration testing is an incremental integration testing technique which begins by testing the top level module and progressively adding in lower level module one by one.
- It provides early working module of the program and so design defects can be found and corrected early.

Top-down Testing

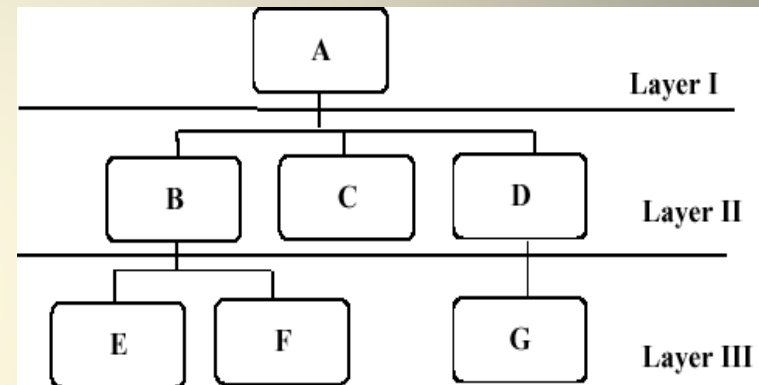


Top-down Testing

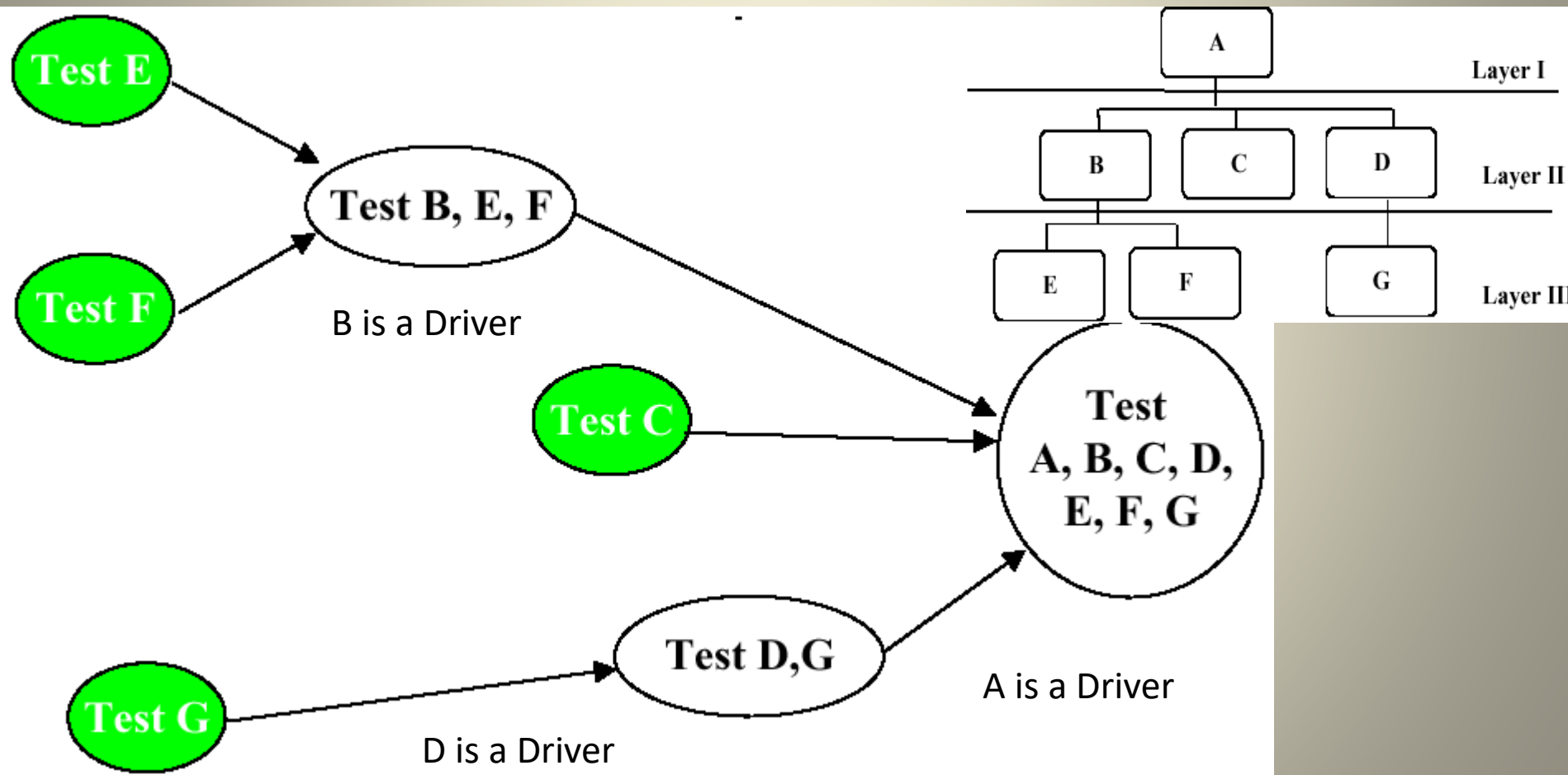


Bottom-up Integration Testing

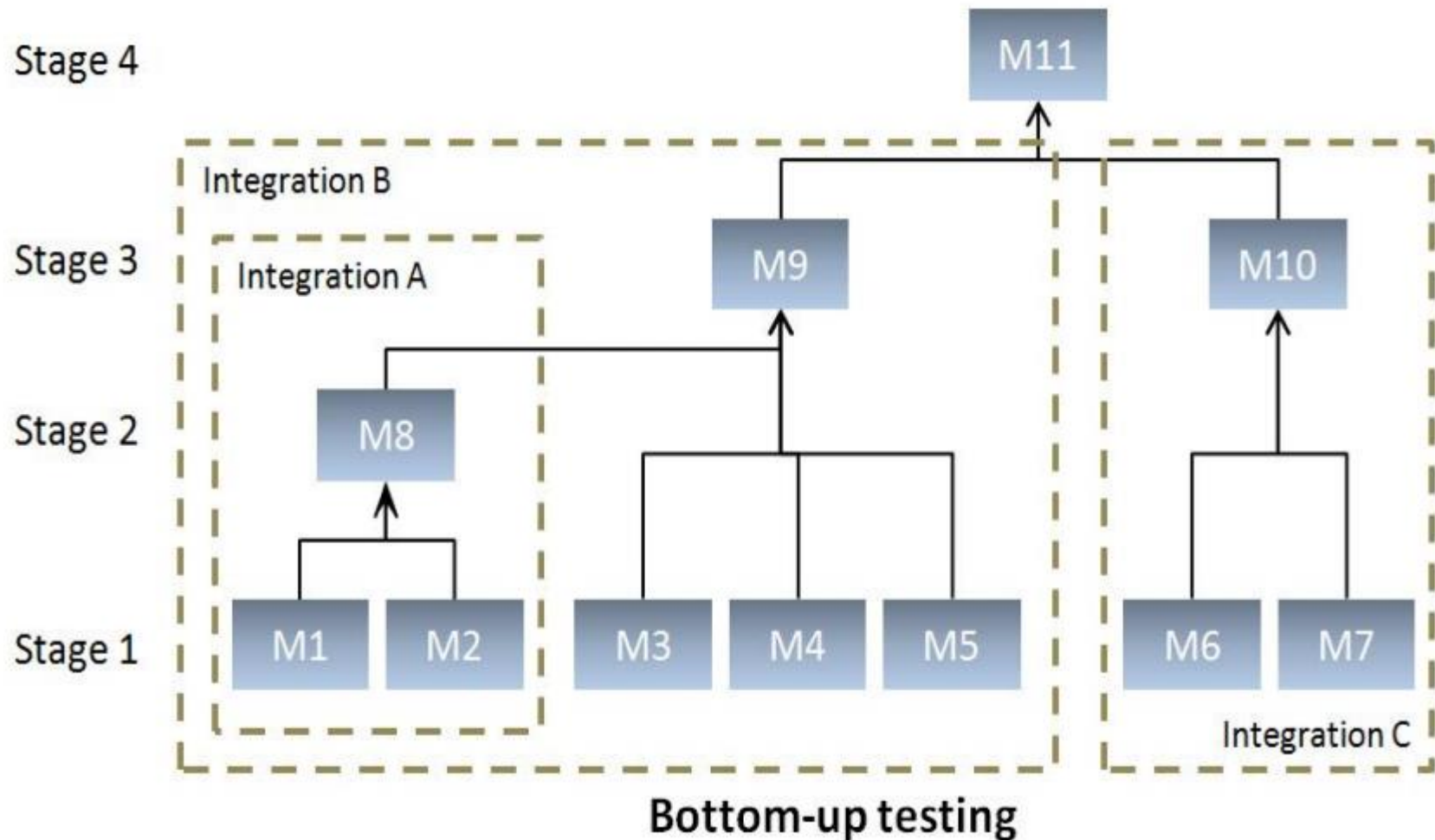
- In bottom up integration testing, module at the lowest level are developed first and other modules which go towards the 'main' program are integrated and tested one at a time.
- In this approach, lower level modules are tested extensively thus make sure that highest used module is tested properly



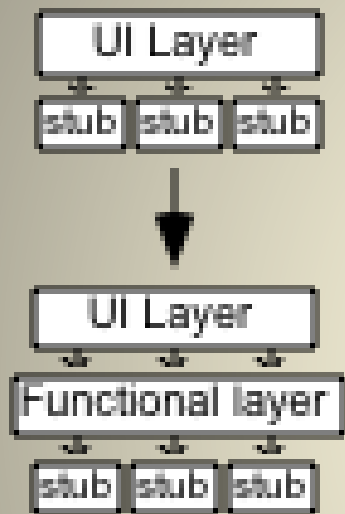
Bottom-up Testing Graphical Representation



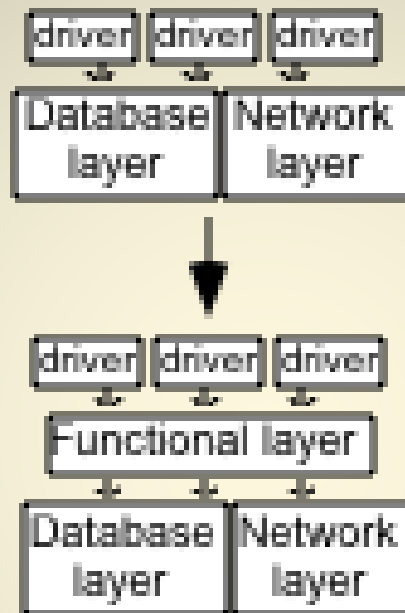
Bottom-up Testing



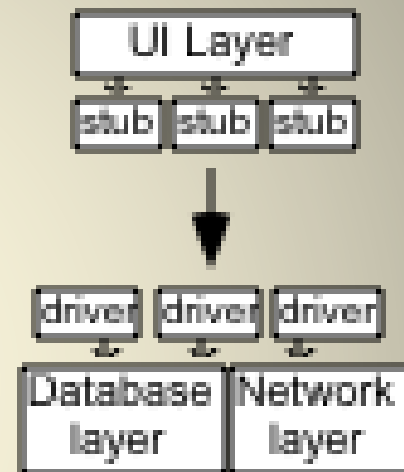
Top-down testing



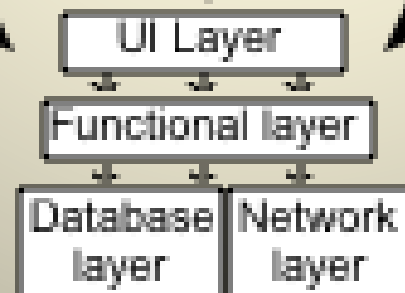
Bottom-up testing



Sandwich testing



Fully
integrated
system



System testing

- System testing is the testing of a complete and fully integrated software product.
- System testing is actually a series of different tests whose sole purpose is to exercise the full computer based system.

Ref: Software Engineering, I. Sommerville, 10th Edition

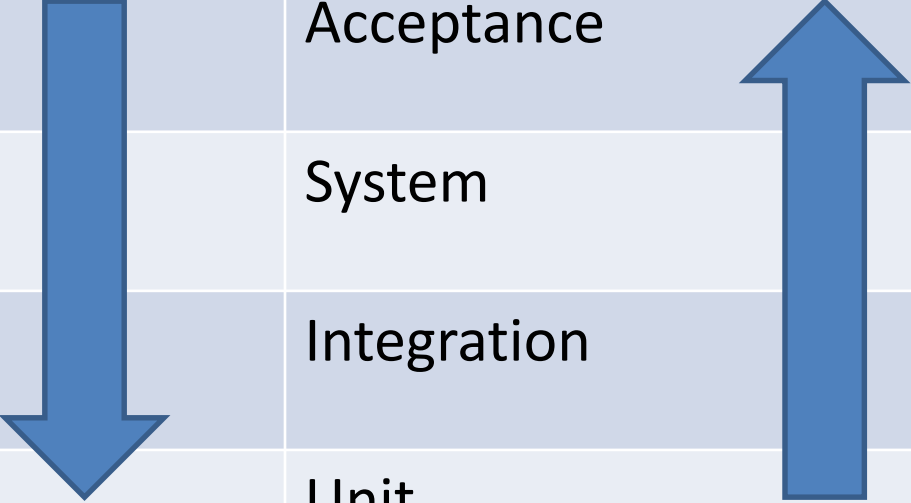
Acceptance testing

- Alpha testing
 - Users of the software work with the development team to test the software at the **developer's site**.
- Beta testing
 - A release of the software is made available to users to allow them to experiment and to raise problems that they discover.
 - tested by users in an **uncontrolled environment**

Ref: Software Engineering, I. Sommerville, 10th Edition

Life cycle model for testing

DEVELOPMENT	TESTING
Requirements	Acceptance
High-level design	System
Detailed design	Integration
Implementation	Unit



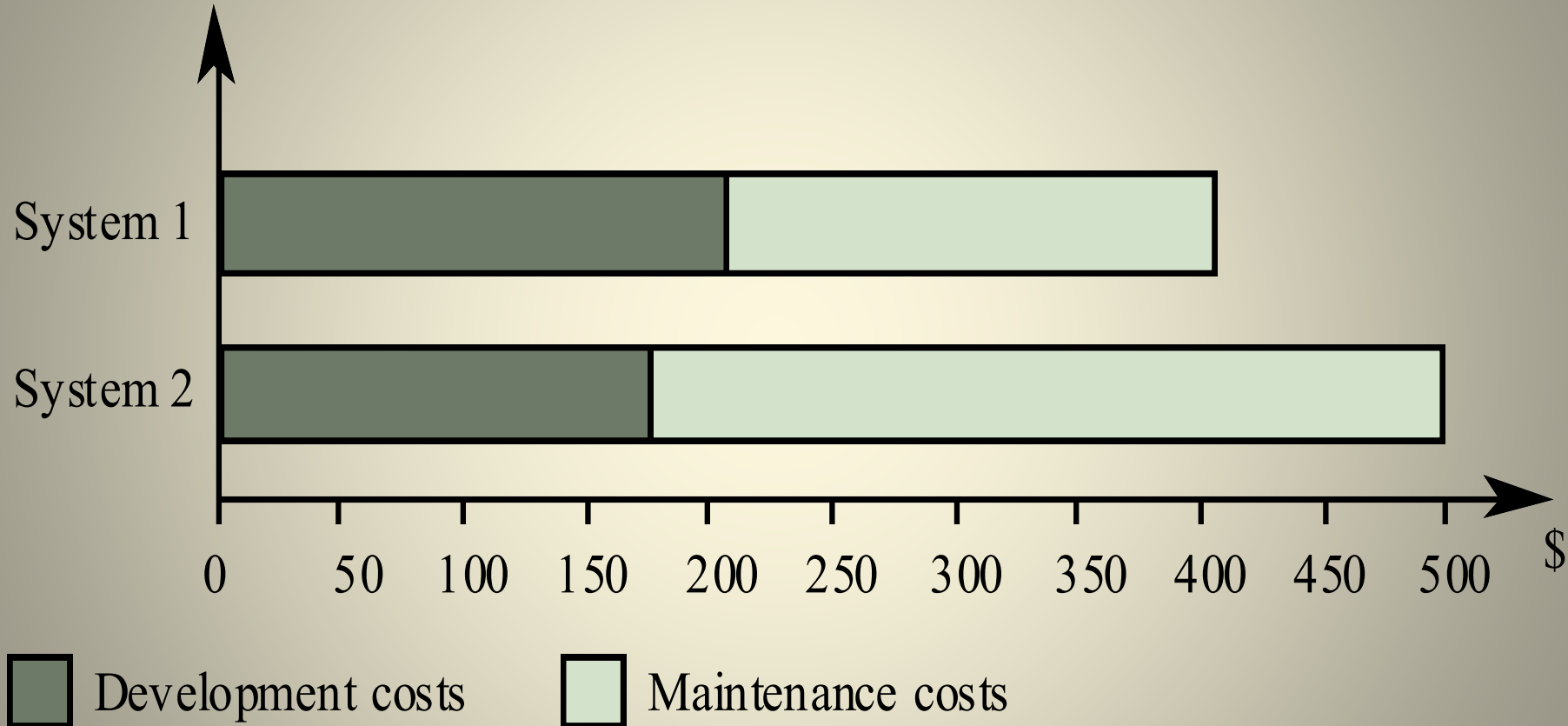
Maintenance

Software Maintenance

“Managing the processes of system change”

- Modifying a program after it has been put into use.
- Software Maintenance results in a service being delivered to the customer.
- The change may be simple changes to correct coding errors, more extensive changes to correct design errors or significant enhancement to correct specification error or accommodate new requirements

Development/Maintenance Costs



Types of Maintenance

- **Perfective maintenance**
 - Changing a system to make it meet its requirements more effective. Adding Functionality.
- **Adaptive maintenance**
 - Maintenance due to changes in environment. New Operating Systems, new hardware
- **Corrective maintenance**
 - Correct newly found bugs

References

- Software Engineering – 10th Edition by Ian Sommerville, Chapter 8