8. Software Testing

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Learning Outcomes

Understand what is software testing and why it is important.

Understand the stages of testing / testing process.

Identify how to select test cases that will geared to discover program defects.

Understand test-first development

Compare and contrast different testing techniques.

8.1. What is Software Testing?

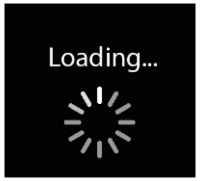
Software testing is a process of executing a program or application with the intent of finding the **software bugs**.

A **software bug**- an error, flaw, failure or fault in a computer **program** or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

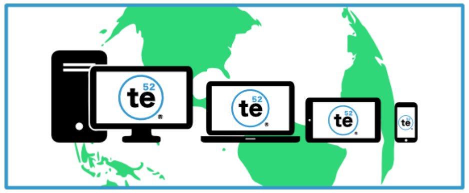
Why Software Testing is so Important?

To ensure it does what it’s supposed to do.

Something that works when one person is using it, may not work when hundreds of people are using it.



There are lots of different devices, browsers, and operating systems out there.



To deliver the best application / high quality product.

Why Software Testing is so Important?

Many More!

- lower maintenance cost
- Increase accuracy, consistency and reliability.
- Increase customer confidence / satisfaction.
- Loss of money
- Loss of time
- Business reputation
- Injury or death
  - some safety-critical systems could result in injuries or deaths if they don’t work properly
    - e.g. flight traffic control software

Clever software can make our lives easier but a glitch can have disastrous consequences

So, Software Testing is…

Intend to show that a program does what it is intended to do and to discover program defects before it is put into use.

Execute a program using artificial / actual data.

Check the results of the test run for errors, anomalies or information about the program's non-functional attributes.

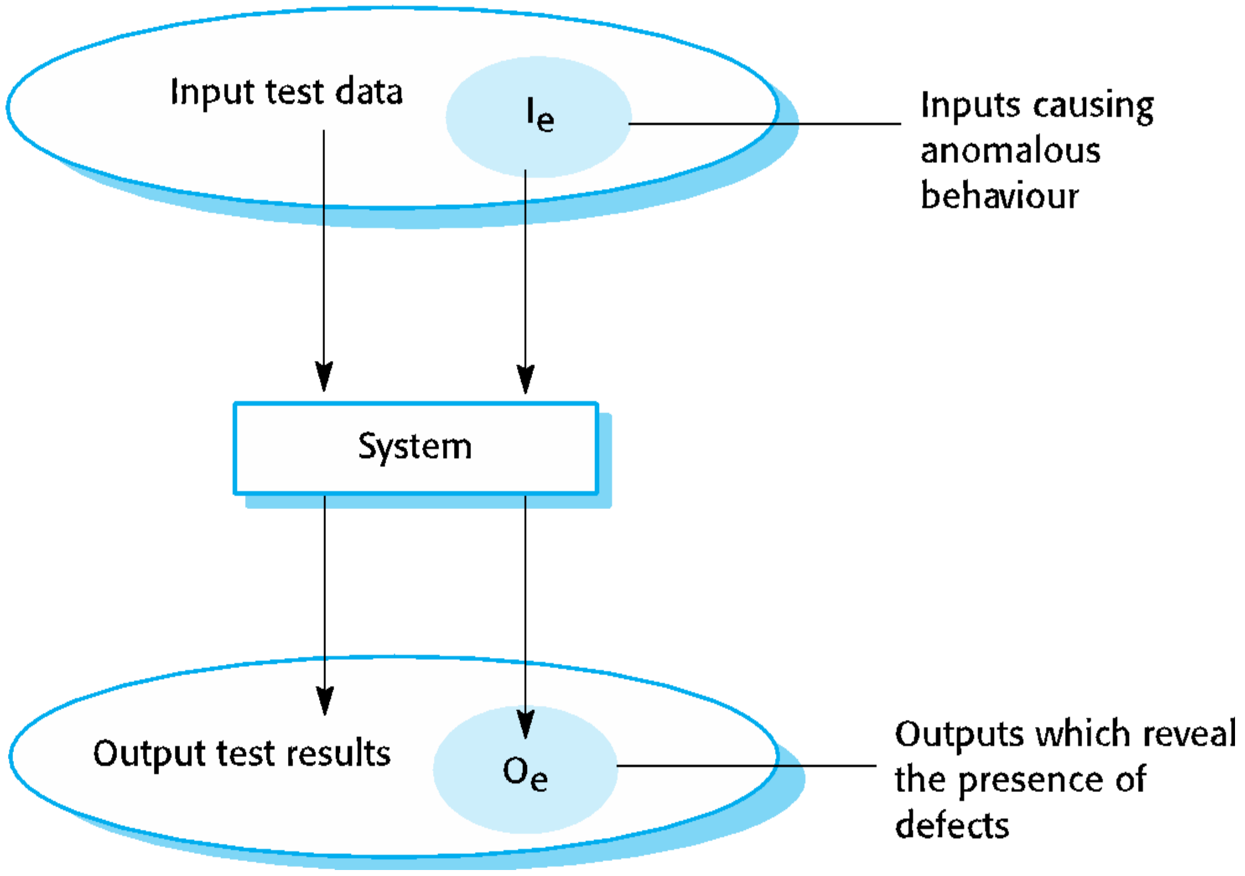
Can reveal the presence of errors NOT their absence.

Main Goals

1. To demonstrate to the developer and the customer that the software meets its requirements.
   - A successful test shows that the system operates as intended.
   - expect the system to perform correctly using a given set of test cases that reflect the system’s expected use.

2. To discover situations in which the behavior of the software is incorrect, undesirable or does not confirm to its specification (Identify Defects)
   - A successful test makes the system perform incorrectly.
   - The test cases are designed to expose defects.

An Input-Output Model of Software Testing


Input test data
I_e
Inputs causing anomalous behaviour
System
Output test results
O_e
Outputs which reveal the presence of defects

V & V in Software Testing

- Software Testing is a broader process of **V**alidation and **V**erification.
- The process of **Validating** and **Verifying** that a software program or application or product meets the business and technical requirements that guided it's design and development.

V & V in Software Testing

- Validation:

"Are we building the right product".

- The software should do what the user really requires.

- Verification:

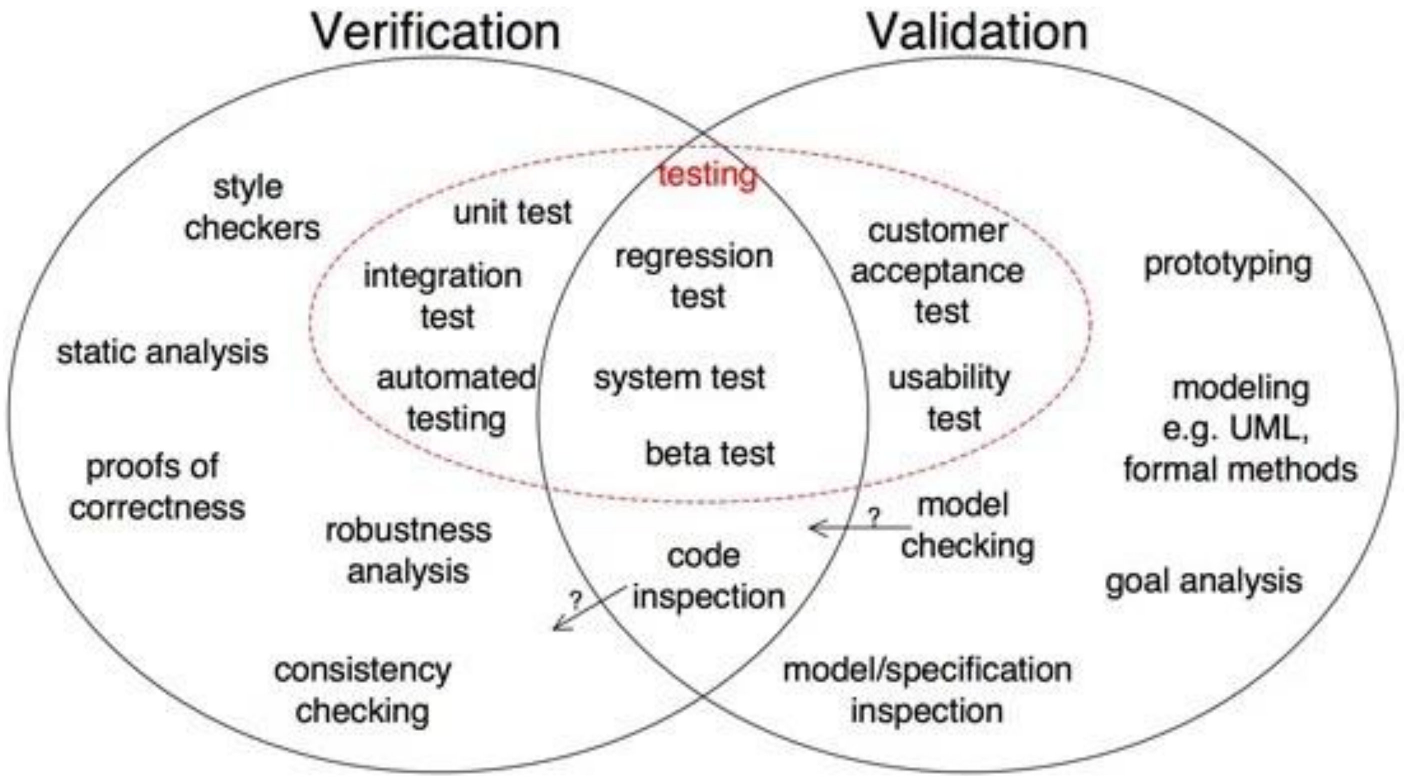
"Are we building the product right".

- The software should conform to its specification.

V & V Goals

- Confidence: system is ‘fit for purpose’.
- Start as soon as requirements become available.
- Continue through all stages of the development process.
- Depends on
    - **Software purpose:** The level of confidence depends on how critical the software is to an organisation.
    - **User expectations:** Users may have low expectations of certain kinds of software.
    - **Marketing environment:** Getting a product to market early may be more important than finding defects in the program.

V & V Techniques

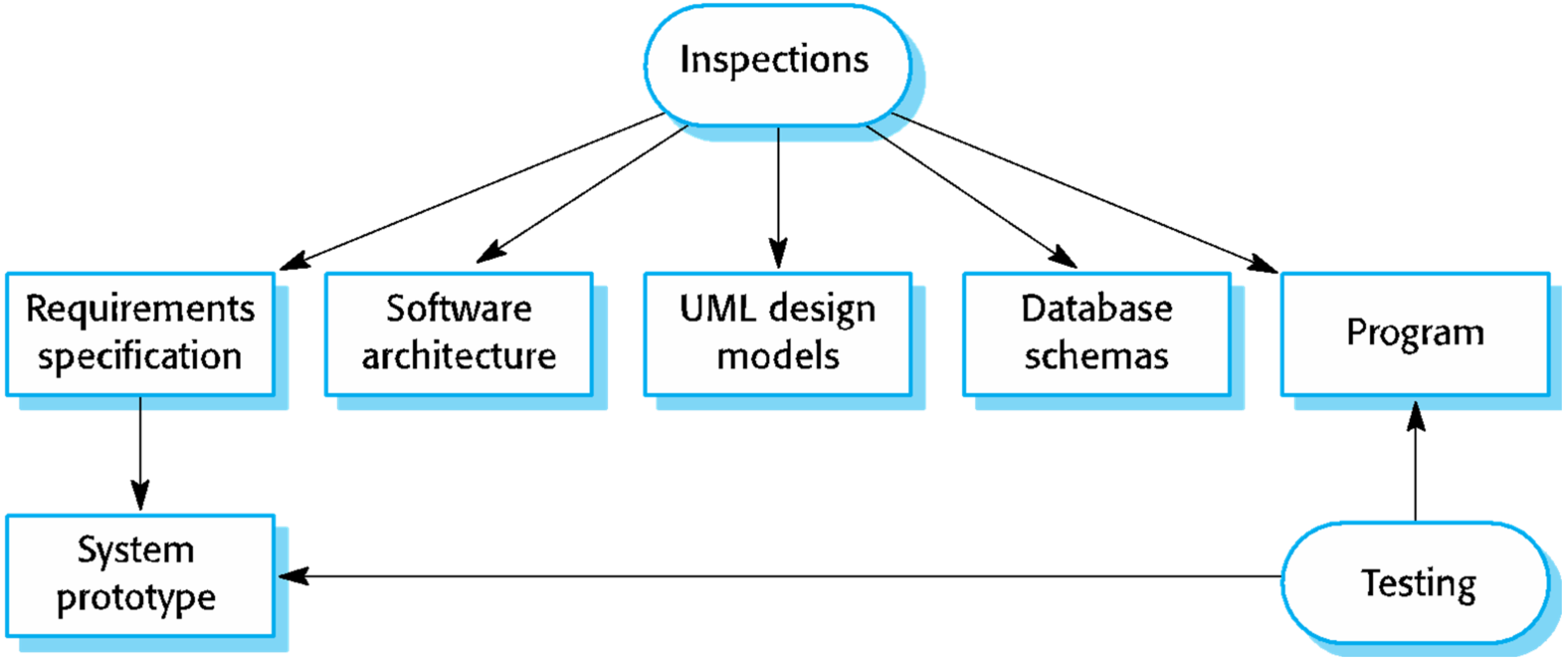


Ref: http://www.easterbrook.ca/steve/2010/11/the-difference-between-verification-and-validation/

V & V Techniques

- **Static** - Software Inspections & Review
  - Concerned with analysis of the static system representation to discover problems.
- **Dynamic** - Defect Testing
  - Concerned with exercising and observing product behavior
  - The system is executed with test data and its operational behavior is observed.

Inspections & Defect Testing



Software Inspections & Reviews

- Applicable to any representation of the system (requirements, design, configuration data, test data, etc.).
- Does not require execution of a system.
- Can be used before implementation.
- An effective technique for discovering program errors.
- Examine the source representation with the aim of discovering anomalies and defects.

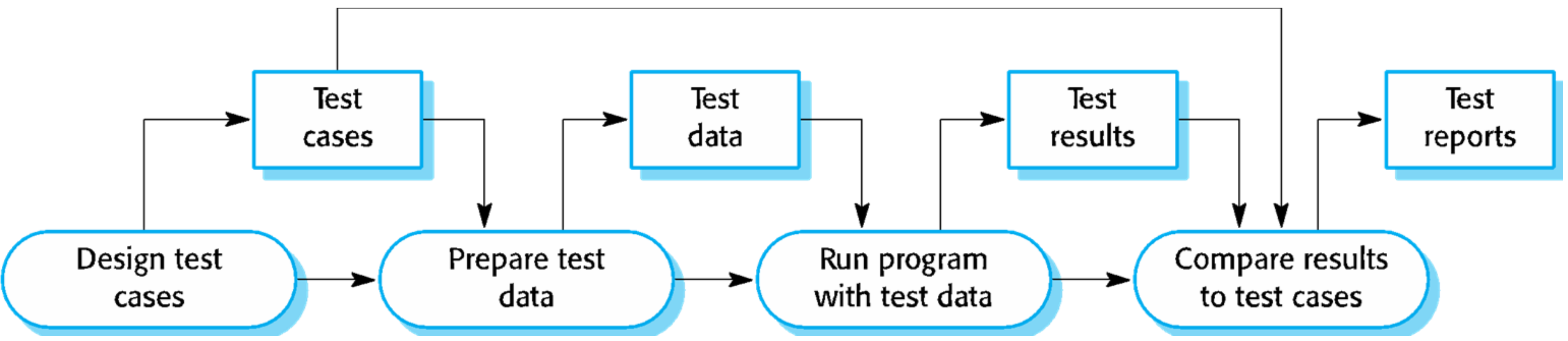
Advantages of Inspections

- Errors may mask (hide) other errors. No need to be concerned with interactions between errors.
- Incomplete versions of a system can be inspected without additional costs.
- Can consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.
- Can look for inefficiencies, inappropriate algorithms and poor programing styles that could make the system difficult to maintain and update.

Inspections vs Defect Testing

- Complementary but not opposing.
- Can be used during the V & V process.
- Inspections can check conformance with a specification but not conformance with the customer’s real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.
- Inspections are NOT good for discovering defects that arise due to unexpected interaction (different parts of a program/ Timing errors / system performance)

Defect Testing Process



What is a Test Case?

- Set of conditions or variables under which a tester will determine whether the system is working correctly or not.
- Corresponds to a use-case.
- Test case comprised of
  - ID
  - Description
  - Input
  - Expected output
- A test case either “Passes” or “Fail”.

Software Defect Testing Stages

- **Development testing:** The system is tested during development to discover bugs and defects.

- **Release testing:** A separate testing team test a complete version of the system before it is released to users.

- **User testing:** Users or potential users of a system, test the system in their own environment.

8.1 Development Testing

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Development Testing

- Includes all testing activities that are carried out by the team developing the system.
  - Unit testing
  - Component testing
  - System testing

8.1.1 Unit Testing

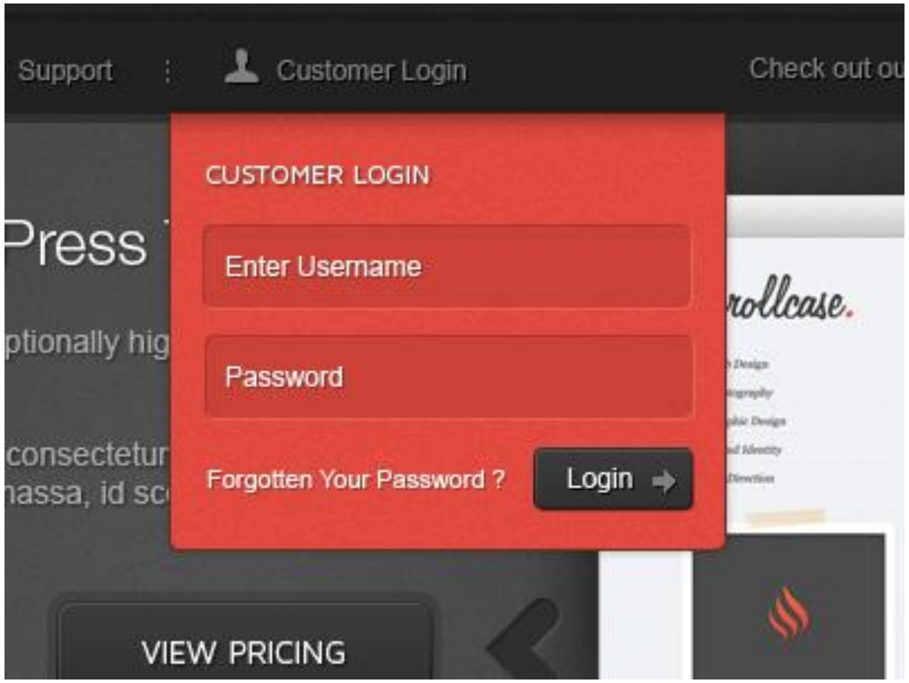
IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

What is Unit Testing

- Isolate each part of the system and show that the individual parts are correct.
  - i.e. Testing individual components in isolation.
- Units may be:
  - Individual functions or methods within an object
  - Object classes with several attributes and methods
  - Composite components with defined interfaces used to access their functionality.

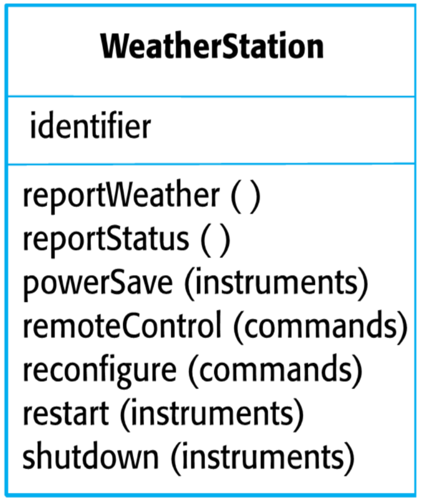
Individual Functions / Methods



Login

Object Class Testing

- Testing all operations associated with an object
- Setting and interrogating all object attributes
- Exercising the object in all possible states.



Automated Testing

- Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.

Choosing Unit Test Cases

- The test cases should show
  - when used as expected, the component that you are testing does what it is supposed to do.
  - If there are defects in the component, these should be revealed by test cases.

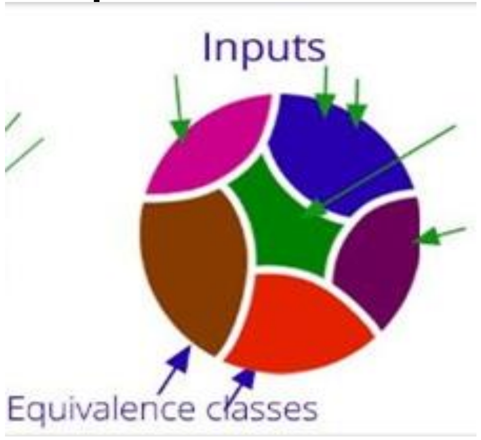
- Two types of unit test cases
  - normal operation of a program and should show that the component works as expected.
  - abnormal inputs to check that these are properly processed and do not crash the component.

Test Cases Selection Strategies

- Partition testing
  - identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- Guideline-based testing
  - Use testing guidelines to choose test cases.
  - Guidelines reflect previous experience of the kinds of errors that programmers often make when developing components.

Partition Testing

- Divide input test data into partitions.
  - Input data and output results often fall into different classes where all members of a class are related
- Assumption: any input within the partition is equivalent - the program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.
- Test each partition.
- Equivalent partitioning.



Equivalence Partitioning: Example 1

- Date Field (Integer values 1 to 31 is accepted)
- Identify the equivalence partitioning.

Invalid

- 0
- -1
- -2
- .
- .
- .

Valid

- 1
- 2
- 3
- …
- 31

Invalid

- 32
- 33
- ..

Equivalence Partitioning: Example 2

- User Name (6 – 10 Characters)
- Identify the equivalence partitioning.

Invalid
- 0 char
- 1 char
- 2 char
- 3 char
- 4 char
- 5 char

Valid
- 6 char
- 7 char
- 8 char
- 9 char
- 10 char

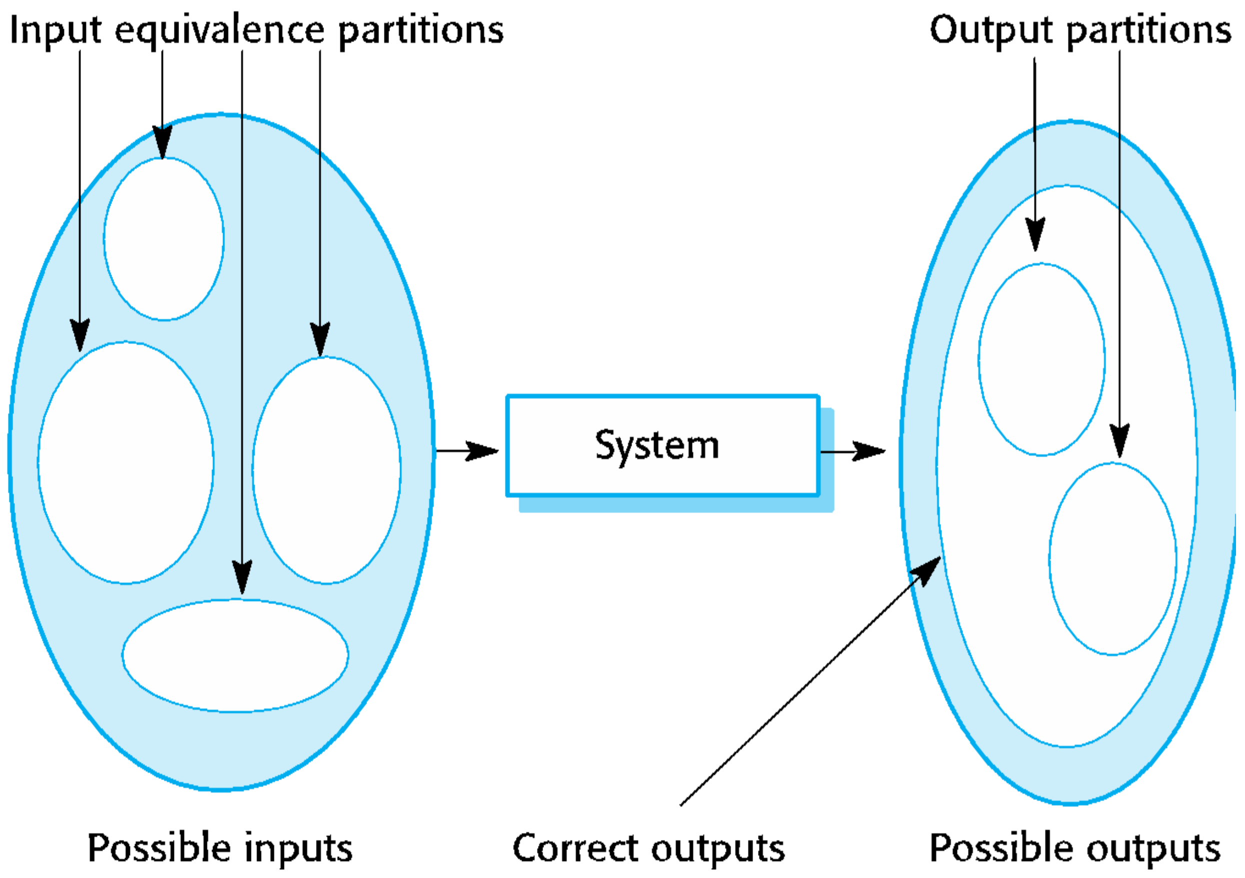
Invalid
- 11 char
- 12 char
- 13 char
- …

Equivalence Partitioning: Example 3

- Age (18 – 80 Years except 60 – 65 Years)
- Identify the equivalence partitioning.

Invalid	Valid	Invalid	Valid	Invalid
•	• 18 Y	• 60 Y	• 66 Y	• 81 Y
• 0 Y	• 19 Y	• 61 Y	• 67 Y	• ......
• 1 Y	•	• 62 Y	•	
•	• 59 Y	• 63 Y	• 80 Y	
• 17 Y		• 64 Y		
		• 65 Y		

Equivalence Partitioning



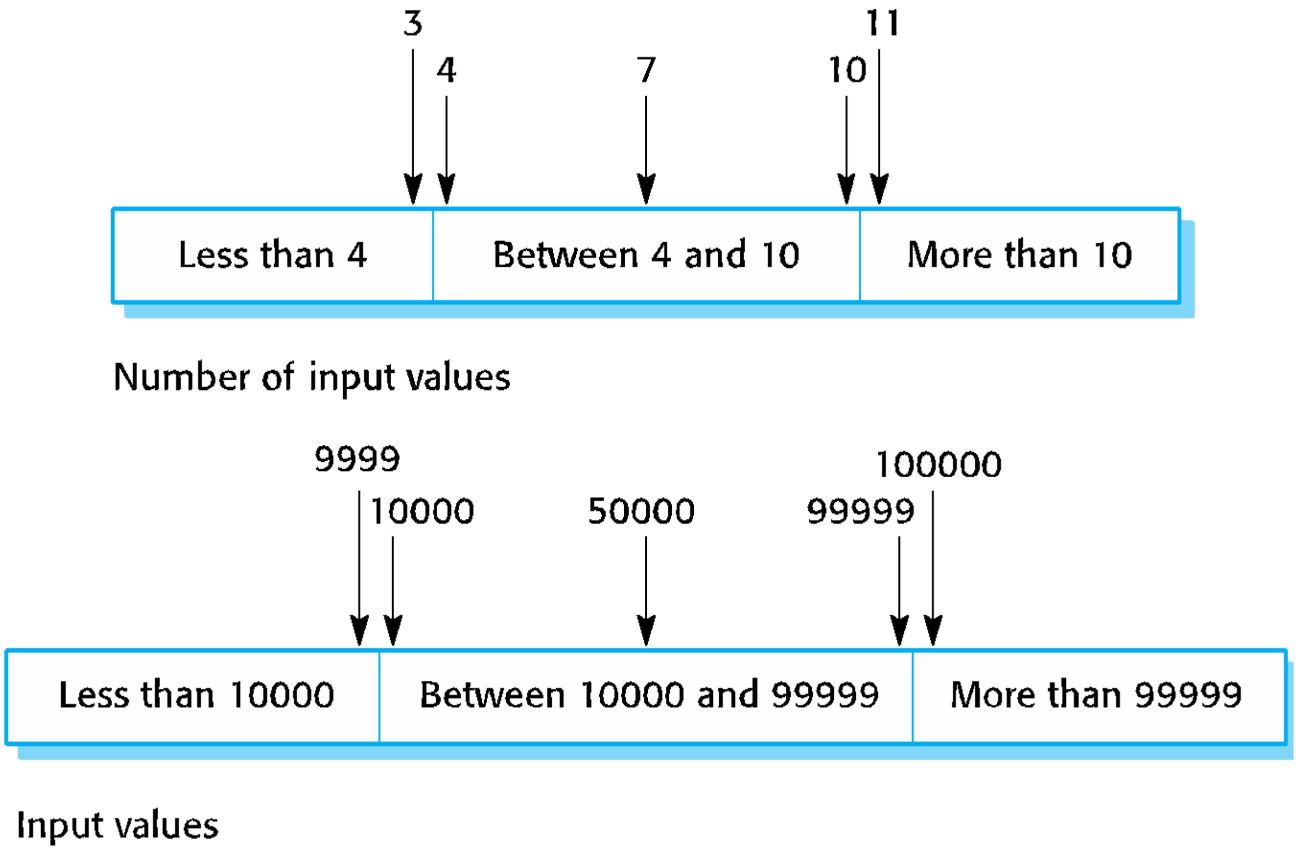
Activity

1. Suppose a program specification states that the program accepts 4 to 10 inputs which are five-digit integers greater than ten thousand (10,000)
   I. Identify the input partitions and possible test input values.

1. You have been asked to test a method called ‘catWhiteSpace’ in a ‘Paragraph’ object that, with in the paragraph, replace sequences of blank characters with a single blank character. Identify testing partitions for this example and derive a set of tests for the ‘catWhiteSpace’ method.

   “The quick brown fox jumped over the lazy dog”

Activity 1: Answer



Activity 2: Answer

- **Testing partitions are:**
  - Strings with only single blank characters.
  - Strings with sequences of blank characters in the middle of the string
  - Strings with sequences of blank characters at the beginning/end of string

- The quick brown fox jumped over the lazy dog(only single blanks)
- The quick brown   fox jumped over the   lazy dog(different numbers of blanks in the sequence)
- The quick brown fox jumped over the lazy dog(1st blank is a sequence)
- The quick brown fox jumped over the lazy dog (Last blank is a sequence)
-   The quick brown fox jumped over the lazy dog(2 blanks at beginning)
-          The quick brown fox jumped over the lazy dog (several blanks at beginning)
- The quick brown fox jumped over the lazy dog   (2 blanks at end)
- The quick brown fox jumped over the lazy dog   (several blanks at end)

Testing Guidelines (sequences)

- Using only a single value
- Use different sizes in different tests.
- Derive tests so that the first, middle and last elements of the sequence are accessed.
- Sequences of zero length.

General Testing Guidelines

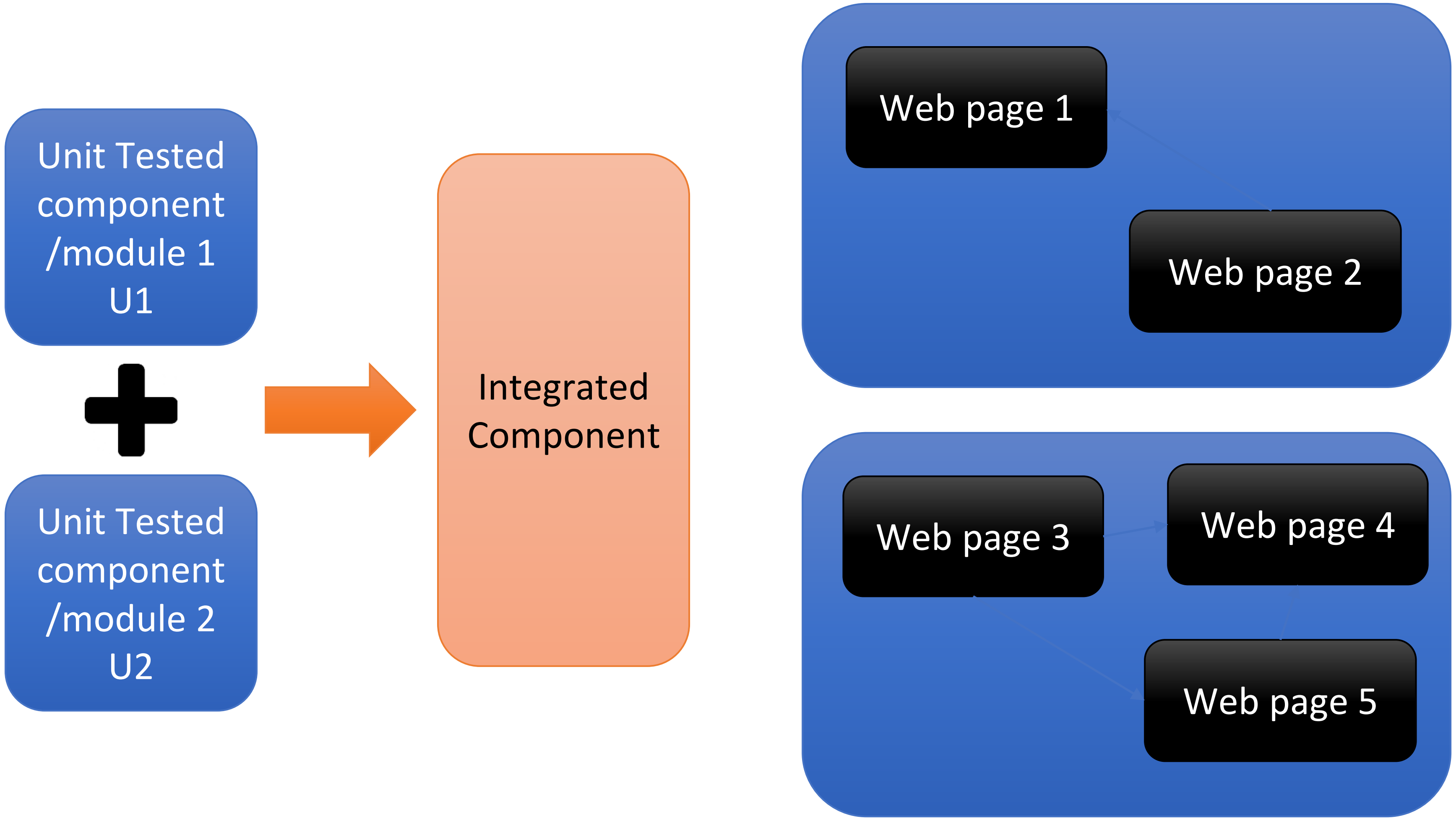
- Choose inputs that force the system to generate all error messages
- Design inputs that cause input buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated
- Force computation results to be too large or too small.

8.1.2 Component Testing

IT2206 - Fundamentals of Software Engineering

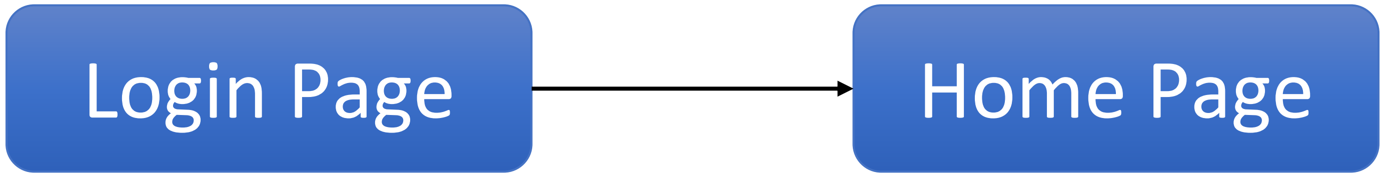
Level I - Semester 2

What is a Component?

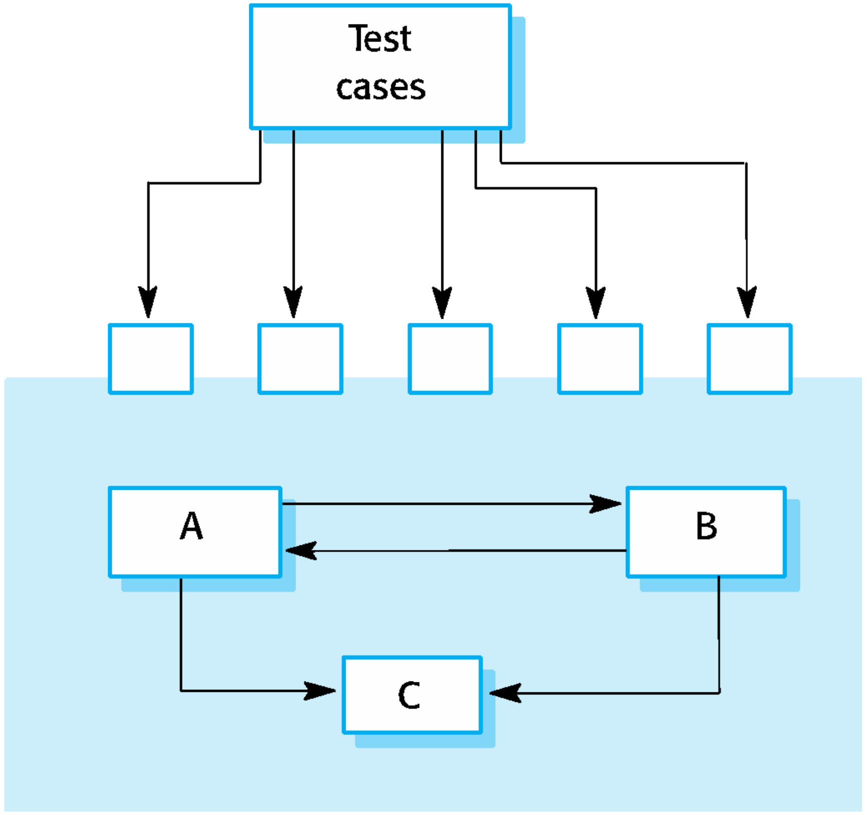


Component Testing

- Composite components are made up of several interacting objects.
- Access the functionality of these objects through the defined component interface.
- Testing composite components should focus on showing that the component interface behaves according to its specification.



Interface Testing



Interface Testing

- Detect faults due to interface errors or invalid assumptions about interfaces.
- Interface types
  - **Parameter interfaces:** Data / function references passed from one component to another.
  - **Shared memory interfaces:** Block of memory is shared between components. Data is placed in memory by one sub-system and retrieved from there by another sub system.
  - **Procedural interfaces:** Sub-system encapsulates a set of procedures to be called by other sub-systems.
  - **Message passing interfaces:** Sub-systems request services from other sub-systems by passing a message.

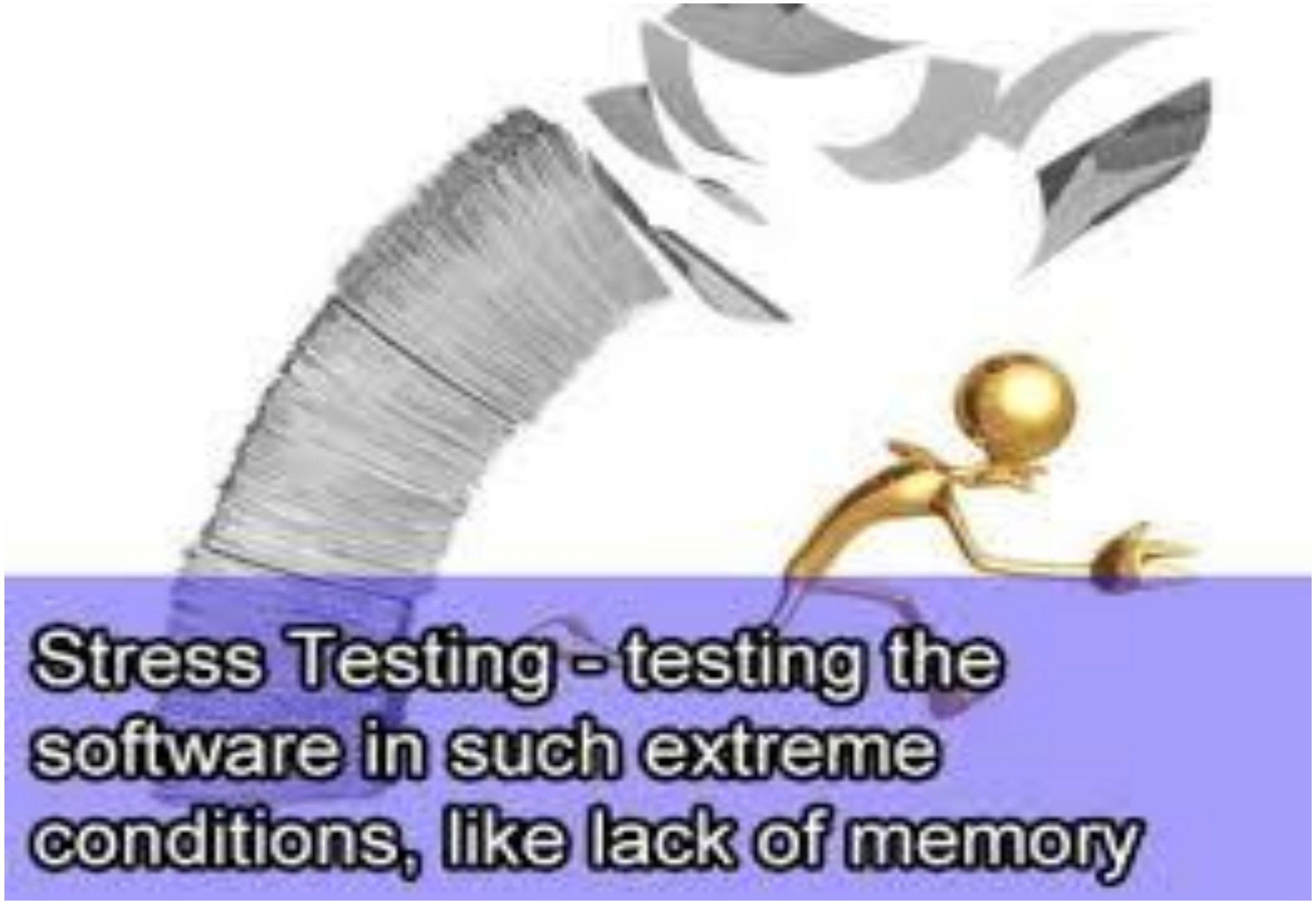
Interface Errors

- **Interface misuse**
  - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.
- **Interface misunderstanding**
  - A calling component embeds assumptions about the behaviour of the called component which are incorrect.
- **Timing errors**
  - The called and the calling component operate at different speeds and out-of-date information is accessed.

Interface Testing Guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- In shared memory systems, vary the order in which components are activated.
- Use **stress testing.**

What is Stress Testing


Stress Testing - testing the software in such extreme conditions, like lack of memory

What is Stress Testing?

- It is a form of **software testing** that is used to determine the stability of a given system.
- A type of **non-functional testing**.
- It involves testing beyond normal operational capacity (breaking point), in order to observe the results.
- It put greater emphasis on robustness, availability, and error handling under a heavy load, rather than correct behavior under normal circumstances.
- The goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space).

8.1.3 System Testing

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

System Testing

- Involves integrating components to create a version of the system and then testing the integrated system.
- The focus is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behavior of a system.

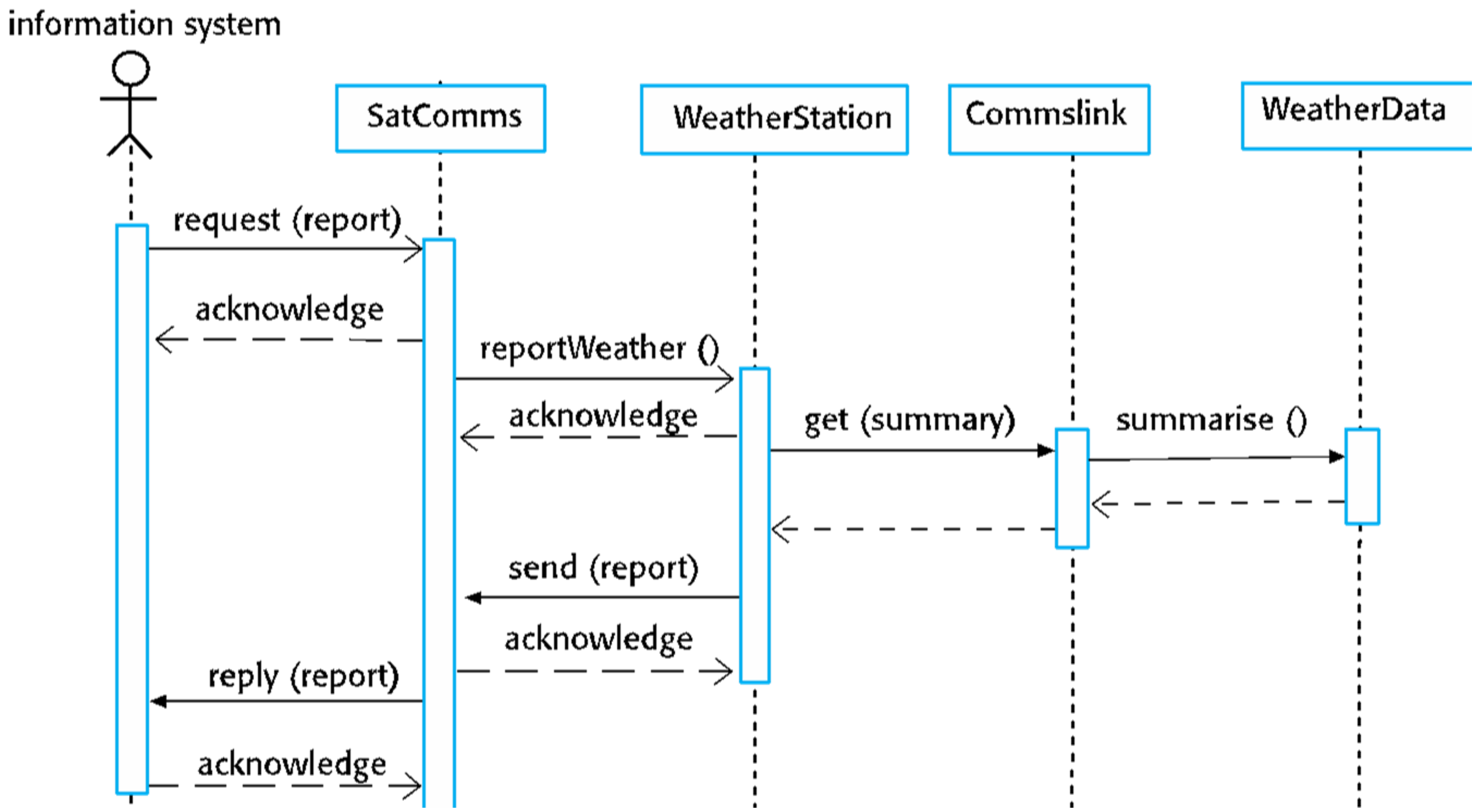
System Testing

- Reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- 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.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

Use-case Testing

- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and interactions that are being tested.

Sequence Diagram: Example



Test Cases derived from Sequence Diagram

- An input of a request for a report should have an associated acknowledgement. A report should ultimately be returned from the request.
  - You should create summarized data that can be used to check that the report is correctly organized.
- An input request for a report to WeatherStation results in a summarized report being generated.
  - Can be tested by creating raw data corresponding to the summary that you have prepared for the test of SatComms and checking that the WeatherStation object correctly produces this summary. This raw data is also used to test the WeatherData object.

Testing Policies

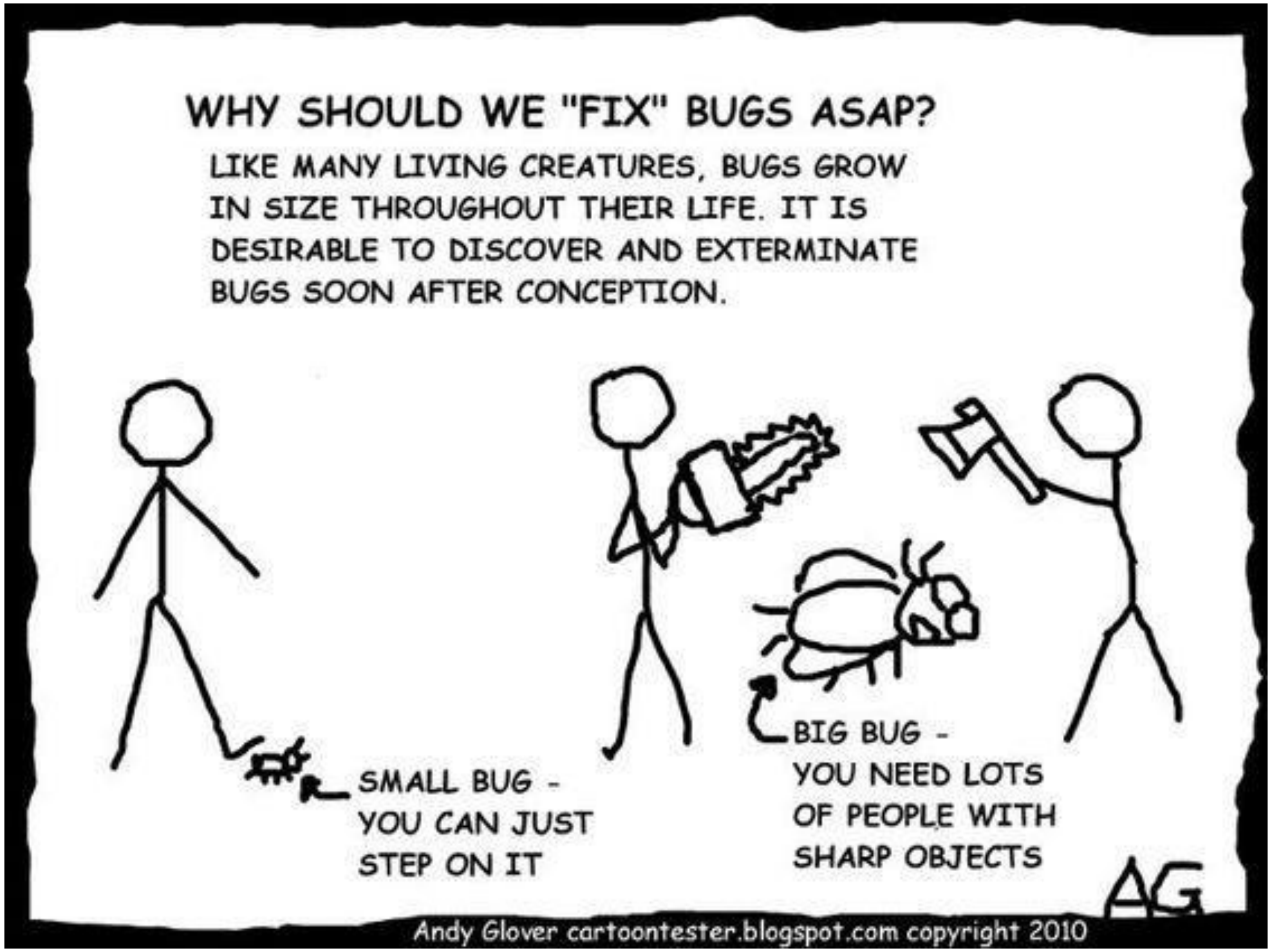
- Difficult to know how much system testing is required.
- Exhaustive system testing is impossible.
- Testing should be based on a subset of possible test cases.
- Software companies should have policies in choosing test cases. Examples of testing policies:
  - All system functions accessed through menus should be tested.
  - Combinations of functions accessed through the same menu must be tested.
  - Where user input is provided, all functions must be tested with both correct and incorrect input.

8.2 Test-driven Development

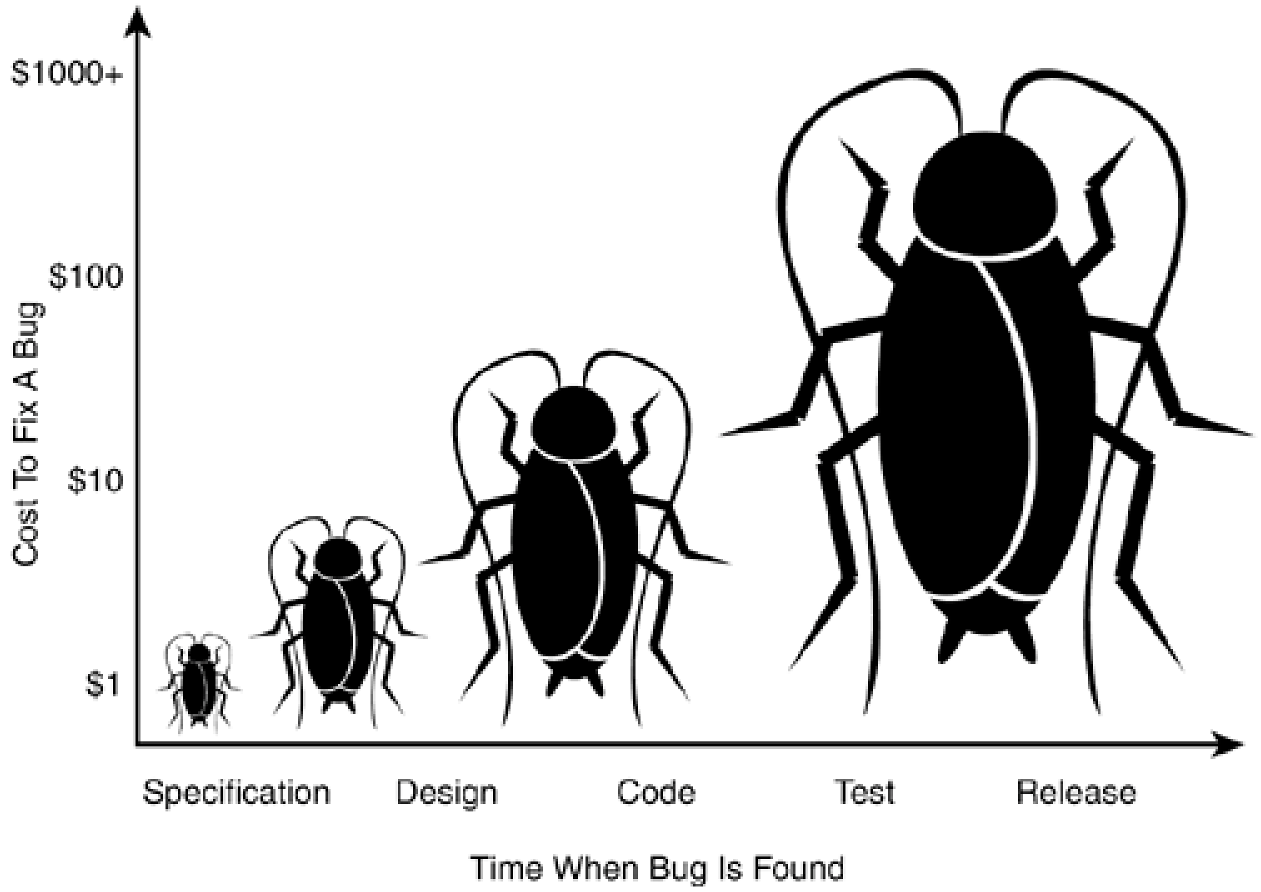
IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Why should you care about TDD?


WHY SHOULD WE "FIX" BUGS ASAP?
LIKE MANY LIVING CREATURES, BUGS GROW IN SIZE THROUGHOUT THEIR LIFE. IT IS DESIRABLE TO DISCOVER AND EXTERMINATE BUGS SOON AFTER CONCEPTION.
SMALL BUG - YOU CAN JUST STEP ON IT
BIG BUG - YOU NEED LOTS OF PEOPLE WITH SHARP OBJECTS
AG
Andy Glover cartoontester.blogspot.com copyright 2010

Why should you care about TDD?

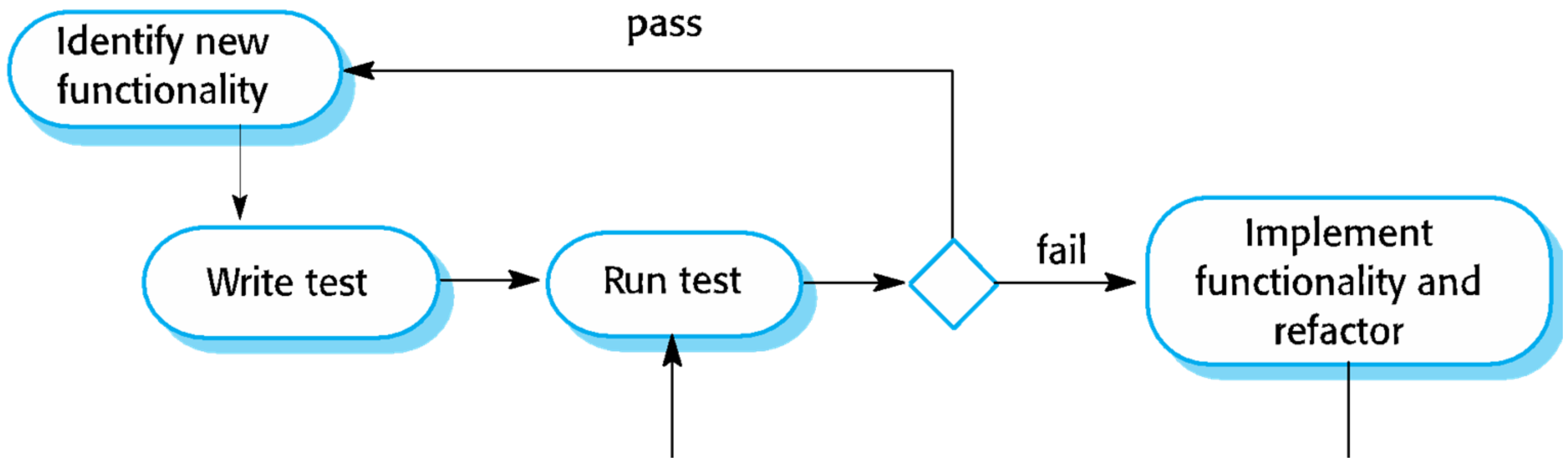


So, we can start with test…

Test-Driven Development

- Tests are written before code.
- Tests should pass and it is a critical driver of development.
- You develop code incrementally, along with a test for that increment.
- You don’t move on to the next increment until the code that you have developed passes its test.
- TDD was introduced as part of agile methods such as Extreme Programming. However, it can also be used in plan-driven development processes.

Test-Driven Development


pass
Identify new functionality
Write test
Run test
fail
Implement functionality and refactor

TDD Process Activities

- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, you have not implemented the functionality so the new test will fail.
- Implement the functionality and re-run the test.
- Once all tests run successfully, you move on to implementing the next chunk of functionality.

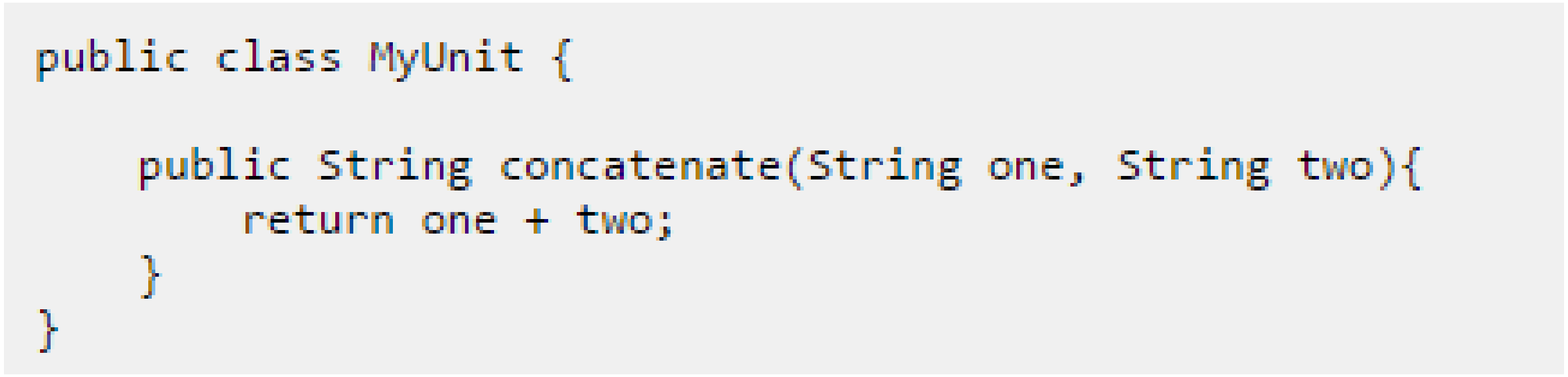
Example

- Suppose you were asked to write a simple function to concatenate two words.
  - Input:
    - One
    - Two
  - Output:
    - OneTwo

Start with a Test

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class MyUnitTest {

    @Test
    public void testConcatenate() {
        MyUnit myUnit = new MyUnit();

        String result = myUnit.concatenate("one", "two");

        assertEquals("onetwo", result);

    }
}
```

Implement Next


```
public class MyUnit {

    public String concatenate(String one, String two){
        return one + two;
    }
}
```

Benefits of Test-Driven Development

- Code coverage
  - Every code segment that you write has at least one associated test so all code written has at least one test.
- Simplified debugging
  - When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified.
- System documentation
  - The tests themselves are a form of documentation that describe what the code should be doing.
- Regression testing
  - A regression test suite is developed incrementally as a program is developed.

What is Regression Testing

To Check Newly Added Functionality
does not affect existing Functionality

Regression Testing

- Regression testing is testing the system to check that changes have not ‘broken’ previously working code.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward.
- All tests are re-run every time a change is made to the program.
- Tests must run ‘successfully’ before the change is committed.

8.3 Release Testing

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Release Testing

- The process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal - Convince that the system is good enough for use.
  - Needs to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- It is usually a **black-box testing** process where tests are only derived from the system specification.

Release Testing and System Testing

- Release testing is a form of system testing.
- Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing).
  - The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Requirements Based Testing

- Requirements-based testing involves examining each requirement and developing a test or tests for it.
- Mentcare system requirements:
  - If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.
  - If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.

Requirements Tests

- Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system.
- Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
- Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
- Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.
- Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

Performance Testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
- Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behavior.

8.4 User Testing

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

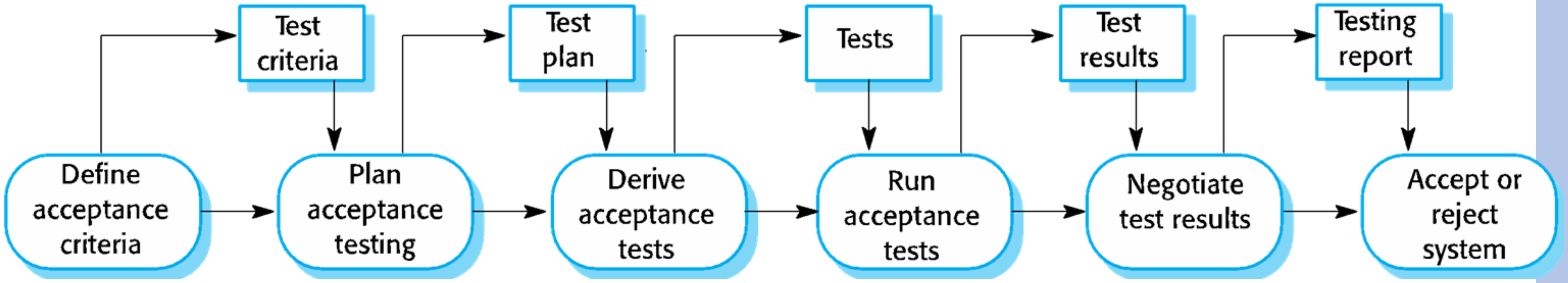
User Testing

- Users or customers provide input and test the system.
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - The reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Types of User 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 with the system developers.
- **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.

The Acceptance Testing Process



Stages in the Acceptance Testing Process

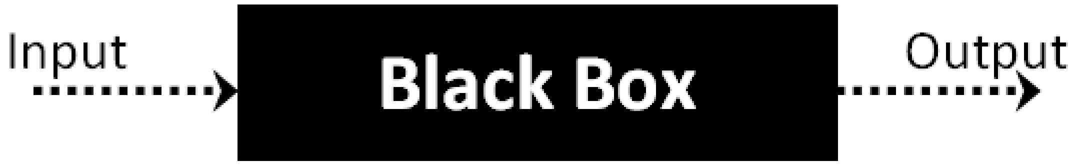
- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system

Agile Methods and Acceptance Testing

- In agile methods, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate acceptance testing process.
- Main problem here is whether or not the embedded user is ‘typical’ and can represent the interests of all system stakeholders.

What is ‘Black Box Testing’

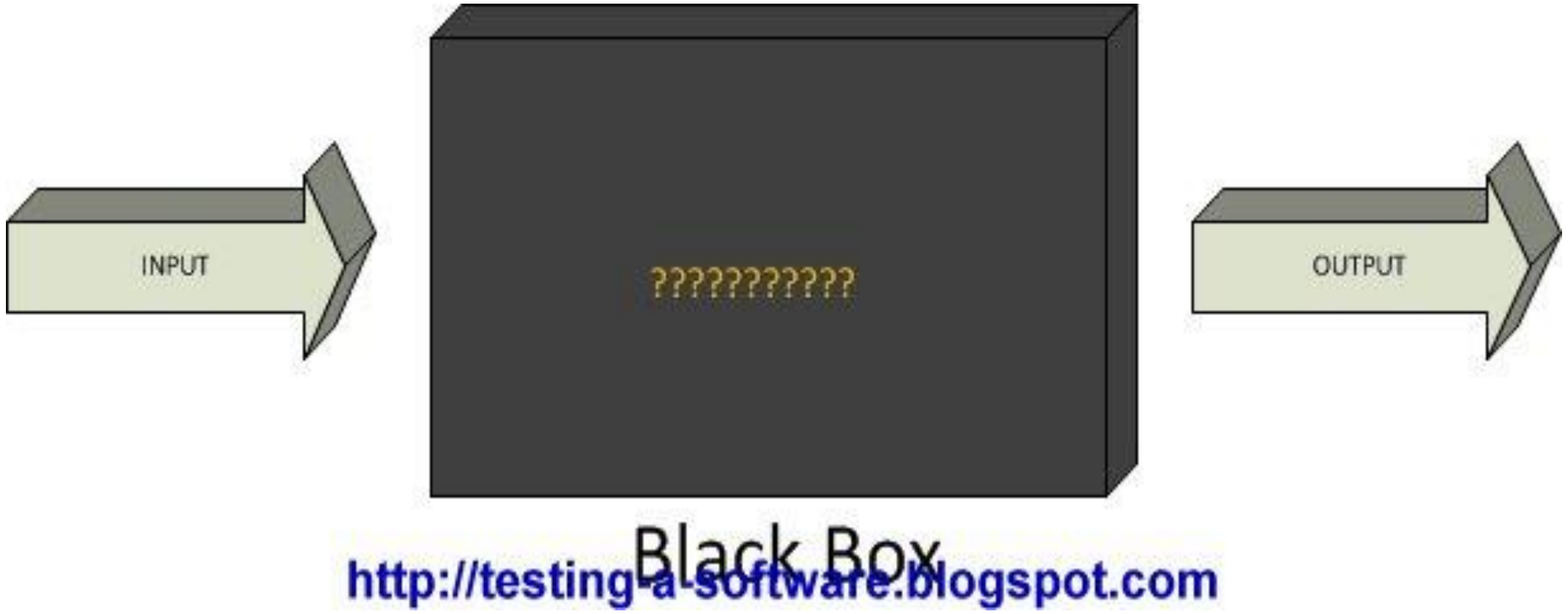
- A method of software testing.
- Examines the functionality of an application without peering into its internal structures or workings.
- Can be applied to every level of software testing:
  - Unit, Integration/Component, System and acceptance.



Black Box: Testing Procedure

- Specific knowledge of the application's code/internal structure and programming knowledge in general is not required.
- The tester is aware of *what* the software is supposed to do but is not aware of *how* it does it.
  - For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of *how* the software produces the output in the first place.

Black Box Testing

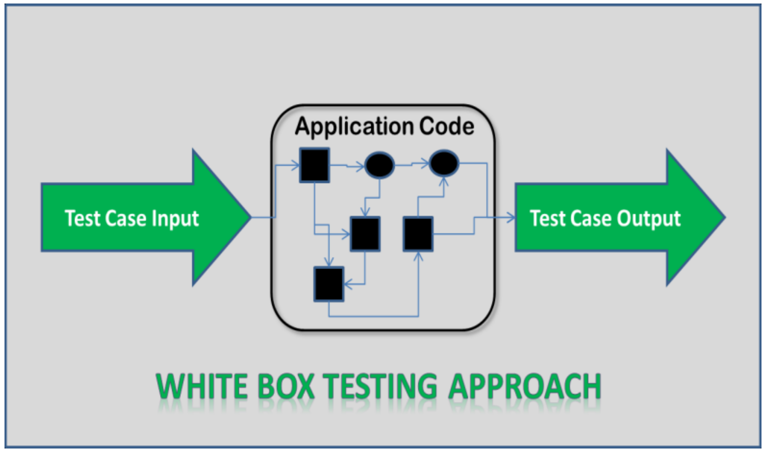


Black Box Testing: Choosing Test Cases

- Test cases are built around specifications and requirements,
- *Both functional* and *non-functional* tests can be applied.
- The test designer selects both valid and invalid inputs and determines the correct output, without any knowledge of the test object's internal structure.

What is White Box Testing

- A method of testing software that tests internal structures or workings of an application.
- Also known as **clear box testing**, **glass box testing**, **transparent box testing**, and **structural testing**
- can be applied at the unit, integration and system levels of the software testing process.



White Box Testing

- Unit testing:
  - to ensure that the code is working as intended, catches any defects early on and prevents errors that can occur later on.
- Integration testing:
  - test the interactions of each interface with each other.
- Regression testing:
  - the use of recycled white-box test cases at the unit and integration testing levels.

Activity

Compare and contrast Black box testing over White box testing.

Summary

- Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.
- Development testing is the responsibility of the software development team. A separate team should be responsible for testing a system before it is released to customers.
- Development testing includes unit testing, in which you test individual objects and methods component testing in which you test related groups of objects and system testing, in which you test partial or complete systems.
- When testing software, you should try to ‘break’ the software by using experience and guidelines to choose types of test case that have been effective in discovering defects in other systems.

Summary

- Wherever possible, you should write automated tests. The tests are embedded in a program that can be run every time a change is made to a system.
- Test-first development is an approach to development where tests are written before the code to be tested.
- Scenario testing involves inventing a typical usage scenario and using this to derive test cases.
- Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operational environment.