- Software Testing

Lecture 1

Topics covered

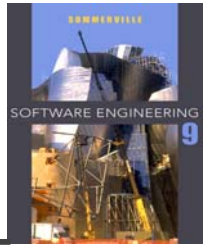
- Development testing
- Test-driven development
- Release testing
- User testing

Software testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.

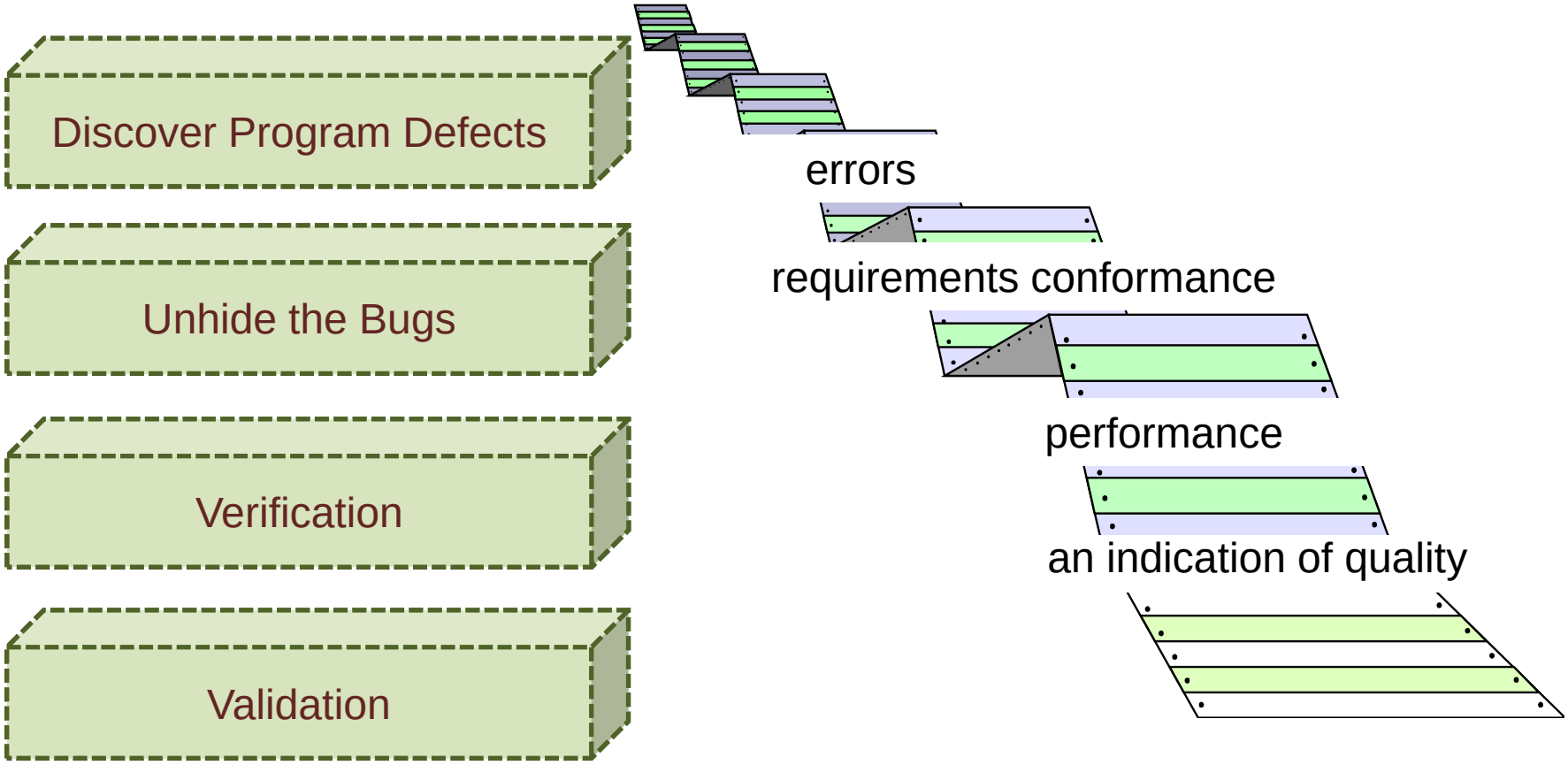
http://en.wikipedia.org/wiki/Software_testing

Program testing

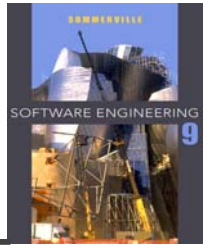


- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program's non-functional attributes.
- Can reveal the presence of errors NOT their absence.
- Testing is part of a more general verification and validation process, which also includes static validation techniques.

Program testing

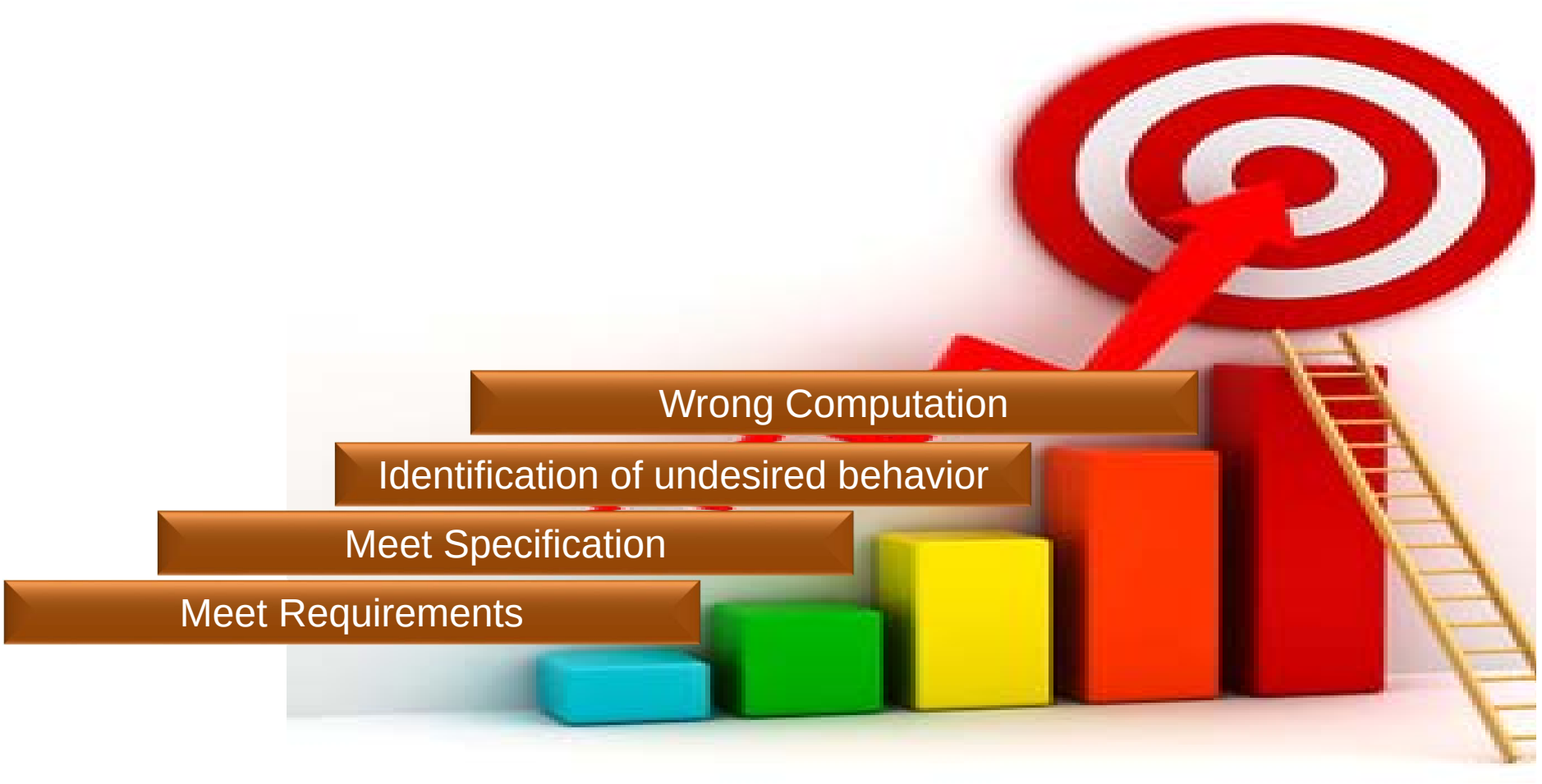


Program testing goals



- To demonstrate to the developer and the customer that the software meets its requirements.
  - For custom software, this means that 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, plus combinations of these features, that will be incorporated in the product release.
- To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
  - Defect testing is concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption.

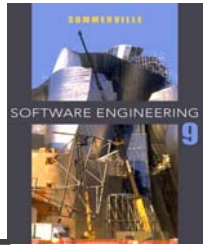
Program testing goals



Validation and defect testing

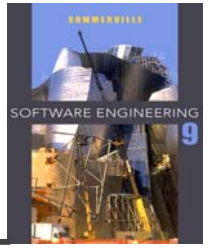
- The first goal leads to validation testing
  - You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
- The second goal leads to defect testing
  - The test cases are designed to expose defects. The test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used.

Validation and defect testing



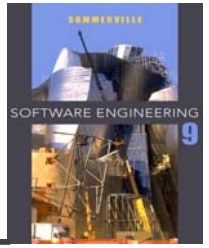
Defect Testing	Wrong Computation
	Identification of undesired behavior
Validation Testing	Meet Specification
	Meet Requirements

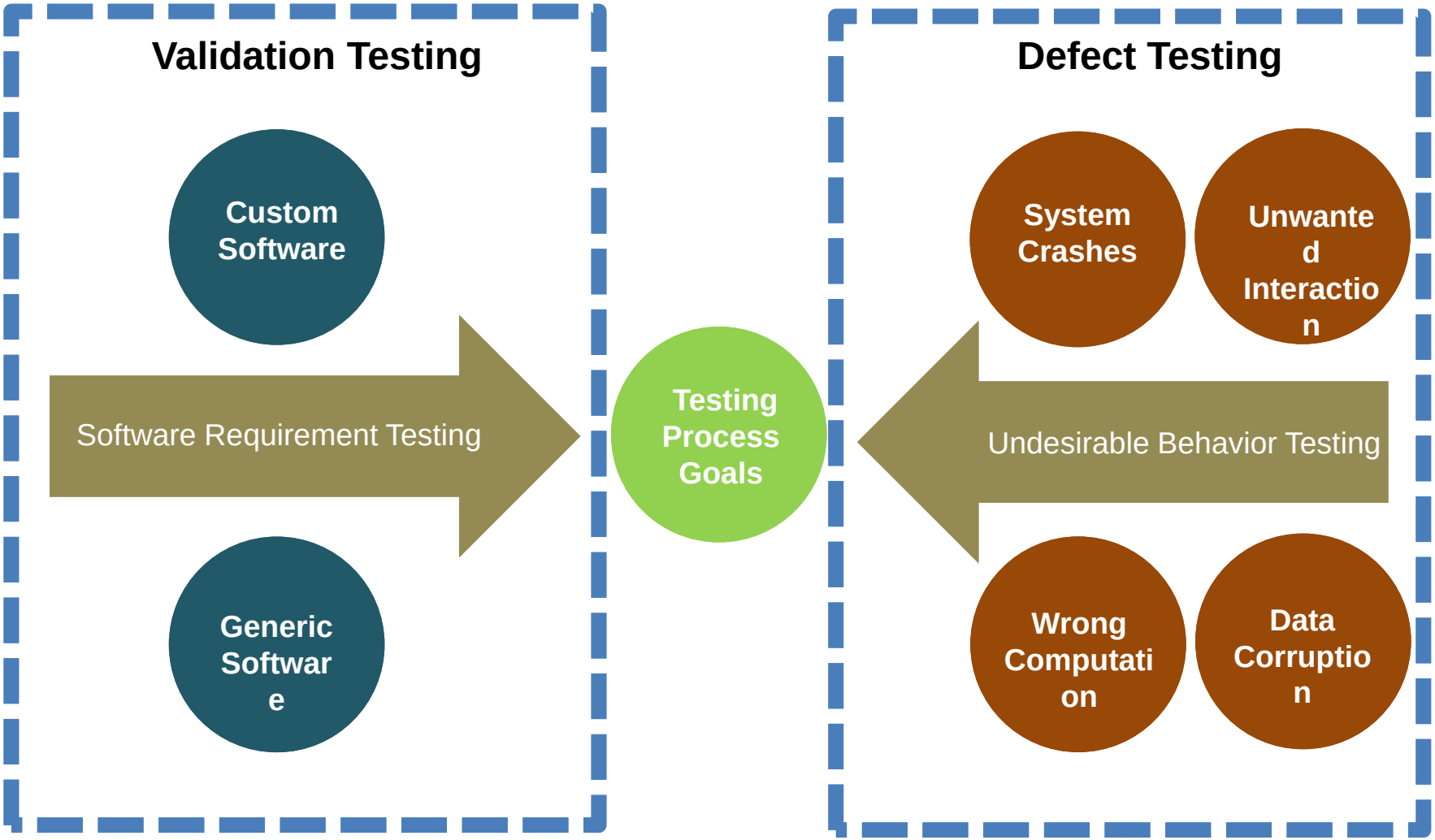
Testing process goals



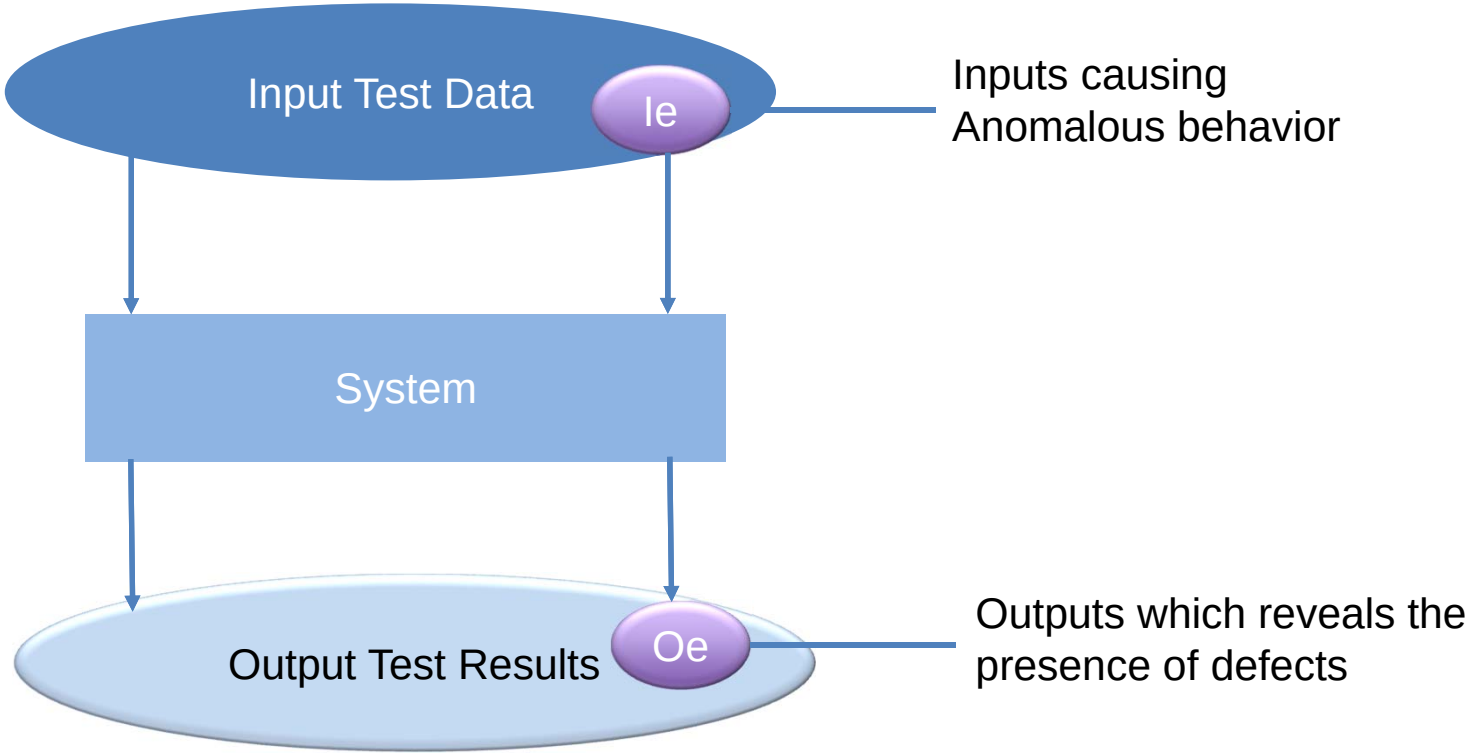
- Validation testing
  - To demonstrate to the developer and the system customer that the software meets its requirements
  - A successful test shows that the system operates as intended.
- Defect testing
  - To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

Testing process goals

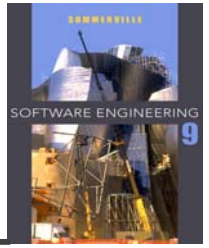




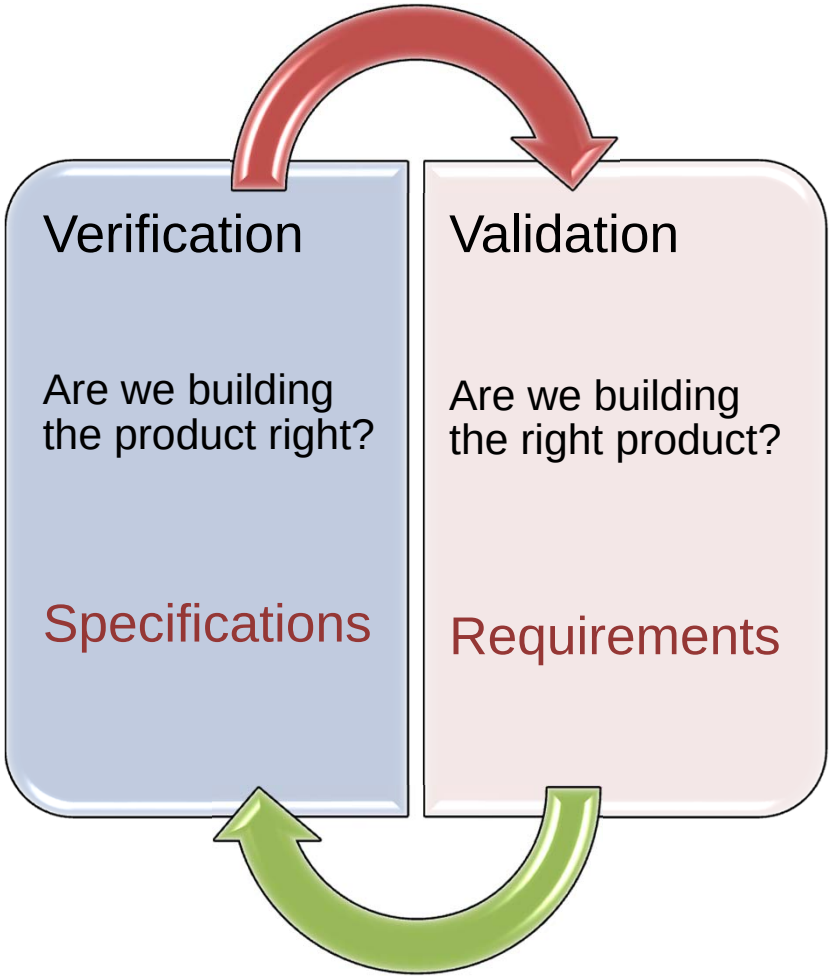
An input-output model of program testing



Verification vs validation

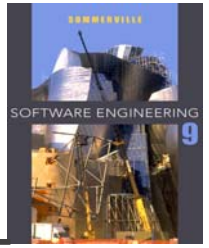


The software should confirm to its specification

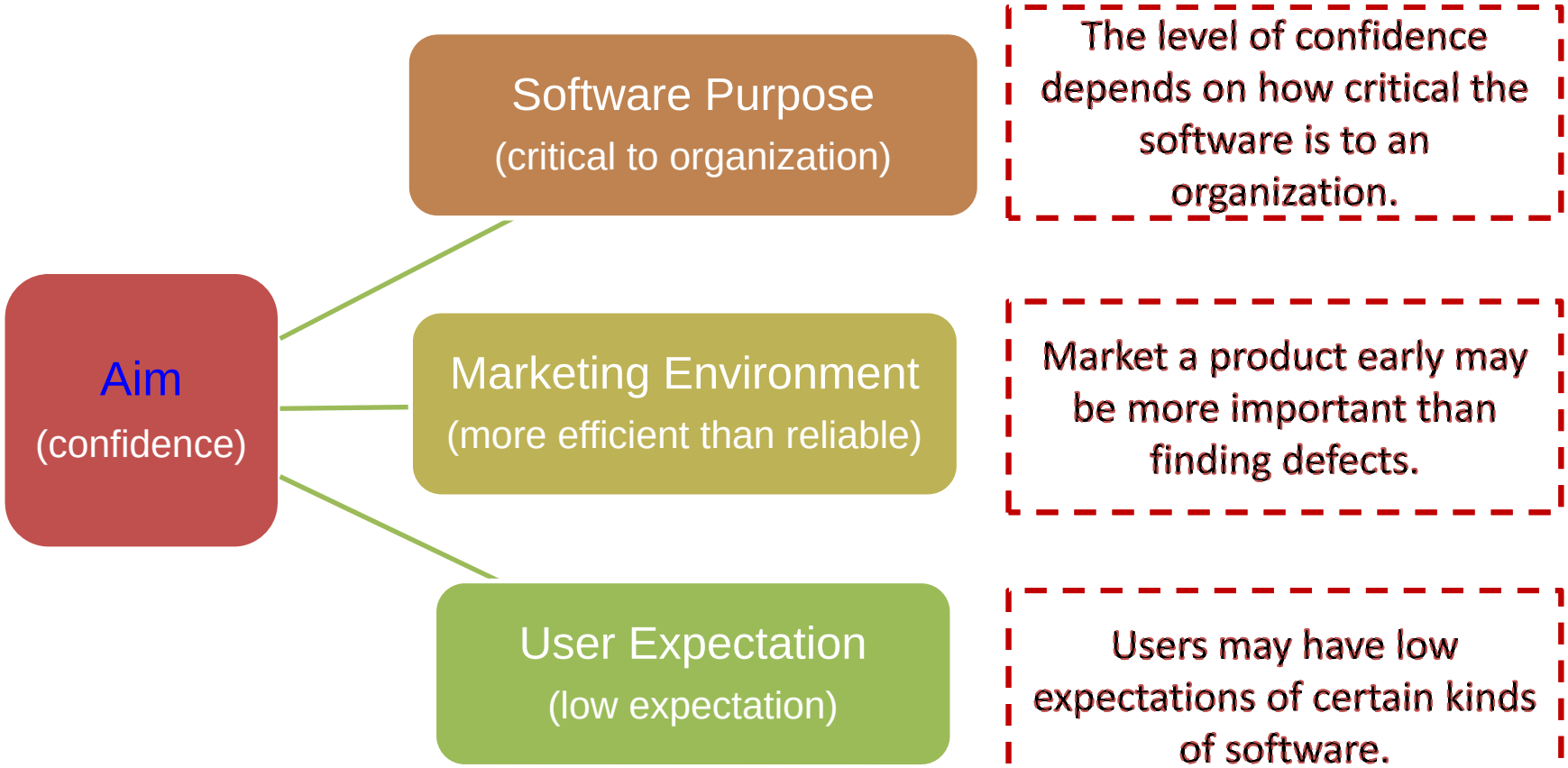


The software should do what the user really requires.

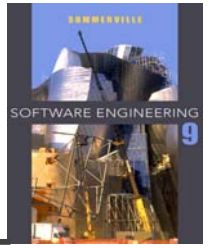
V & V confidence



- Aim of V & V is to establish confidence that the system is ‘fit for purpose’.



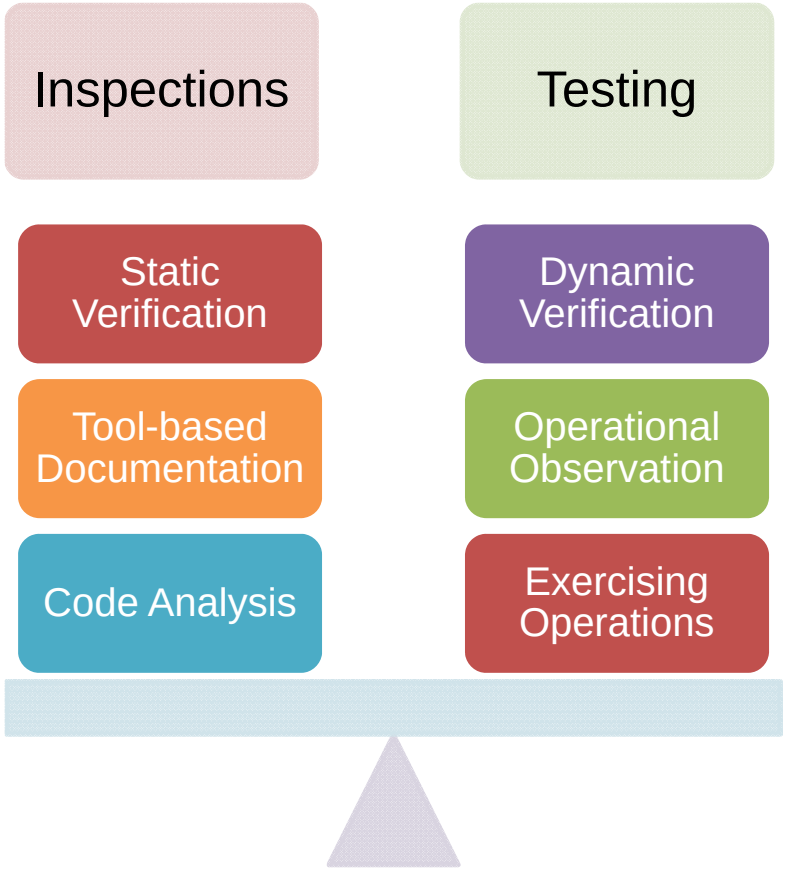
Inspections and testing



Software inspections:

Concerned with analysis of the static system representation to discover problems (static verification)

May be supplement by tool-based document and code analysis

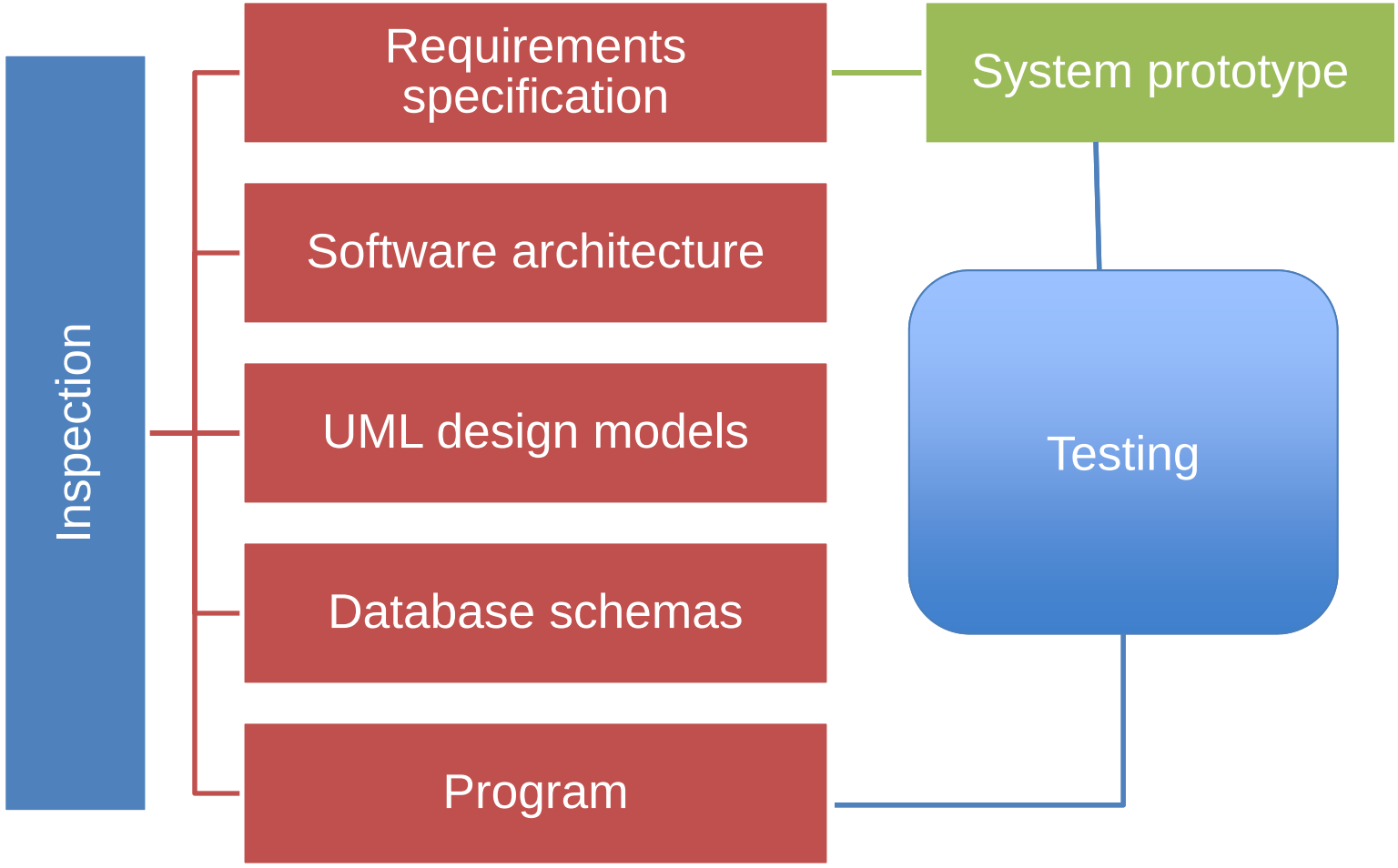


Software testing:

Concerned with exercising and observing product behaviour (dynamic verification)

The system is executed with test data and its operational behaviour is observed.

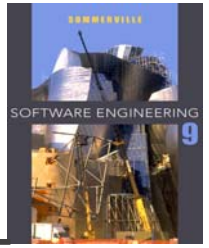
Inspections and testing



Software inspections

- These involve people examining the source representation with the aim of discovering anomalies and defects.
- Inspections not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.

Advantages of inspections



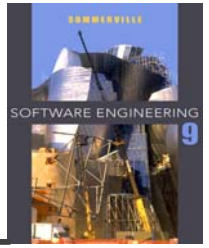
- During testing, errors can mask (hide) other errors. Because inspection is a static process, you don't have to be concerned with interactions between errors.
- Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.
- As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

Advantages of inspections

Advantages

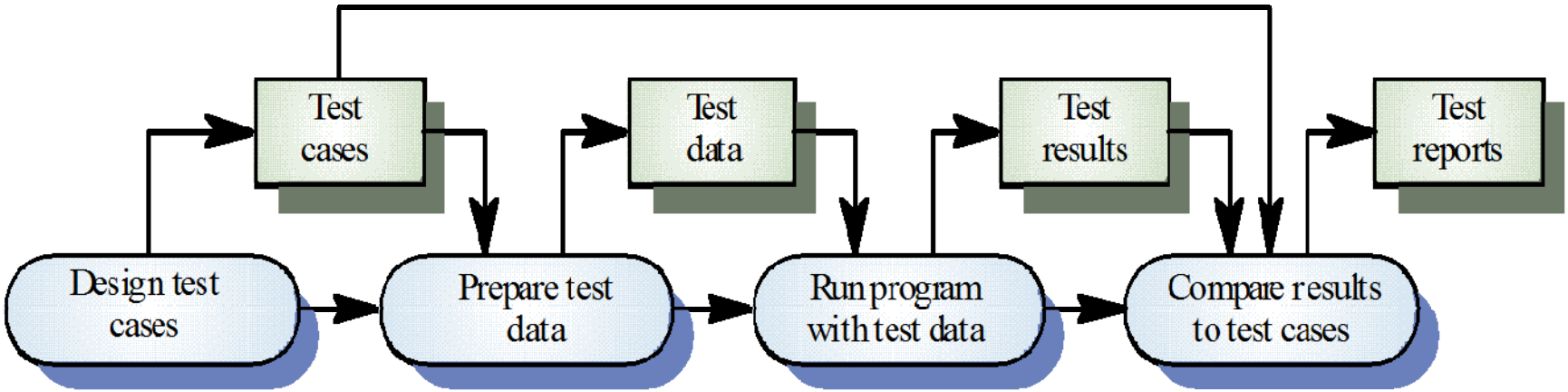
- Broader quality attributes
  (standards, portability, maintainability)
- Incomplete Versions
  (no additional components)
- Interconnected errors
  (errors can't mask other errors)

Inspections and testing

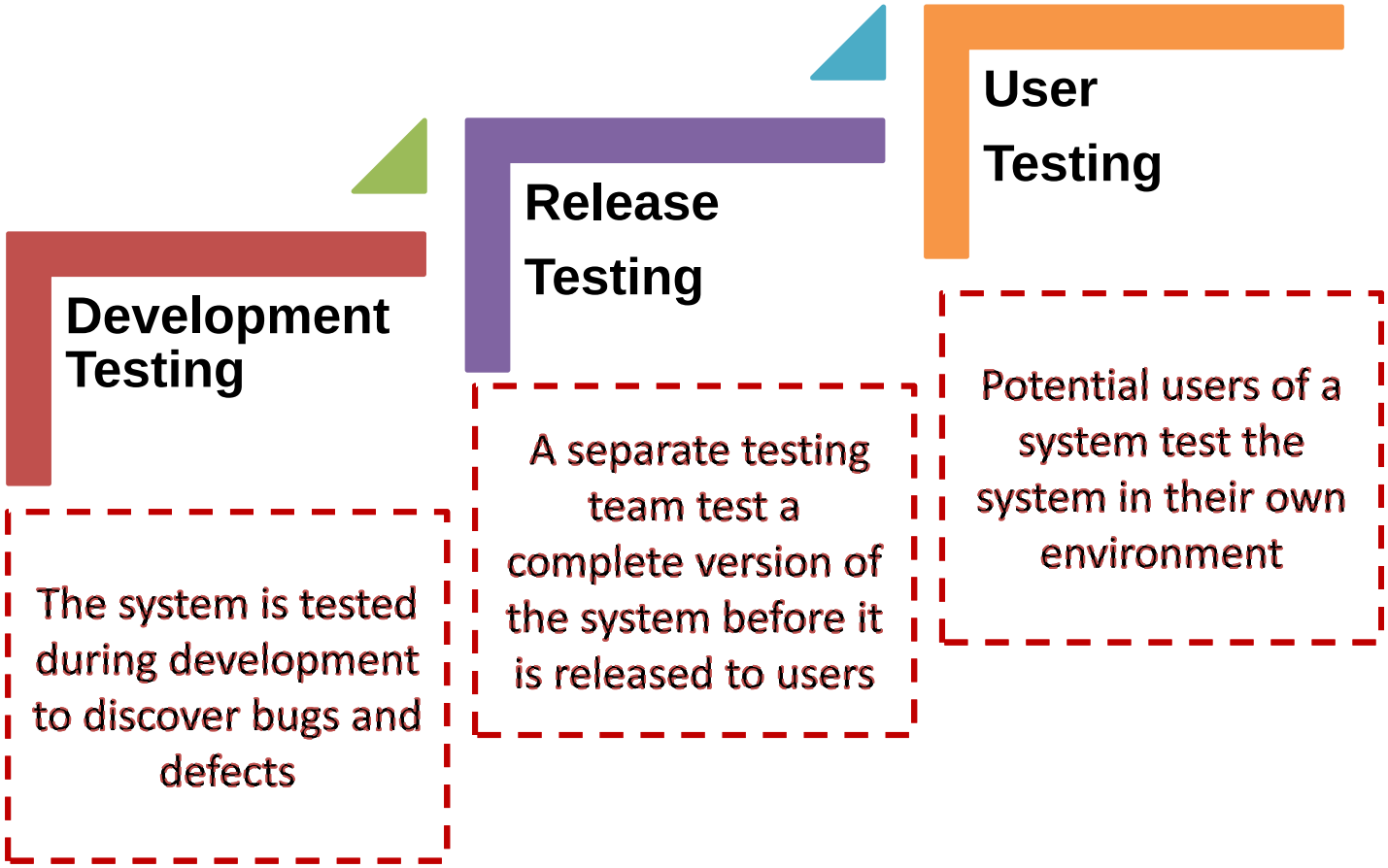


- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

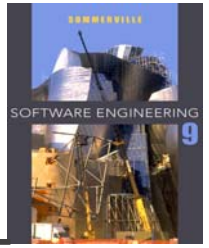
A model of the software testing process



Stages of testing

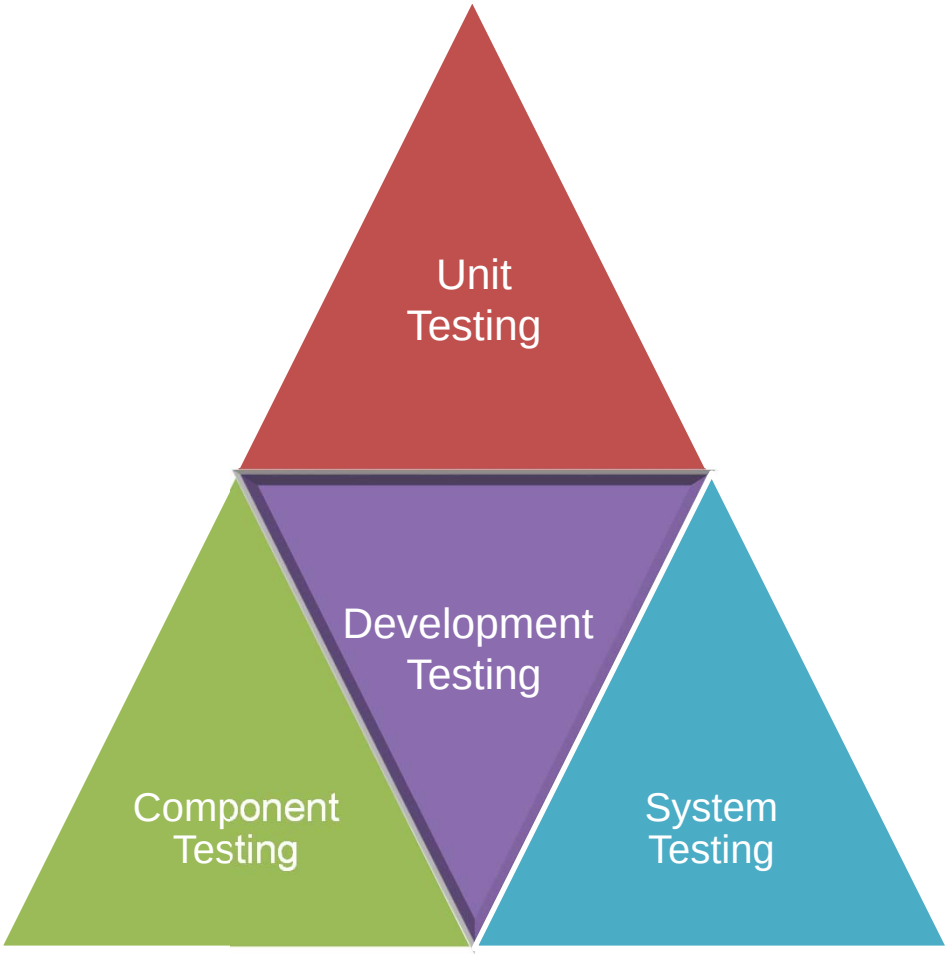


Development testing



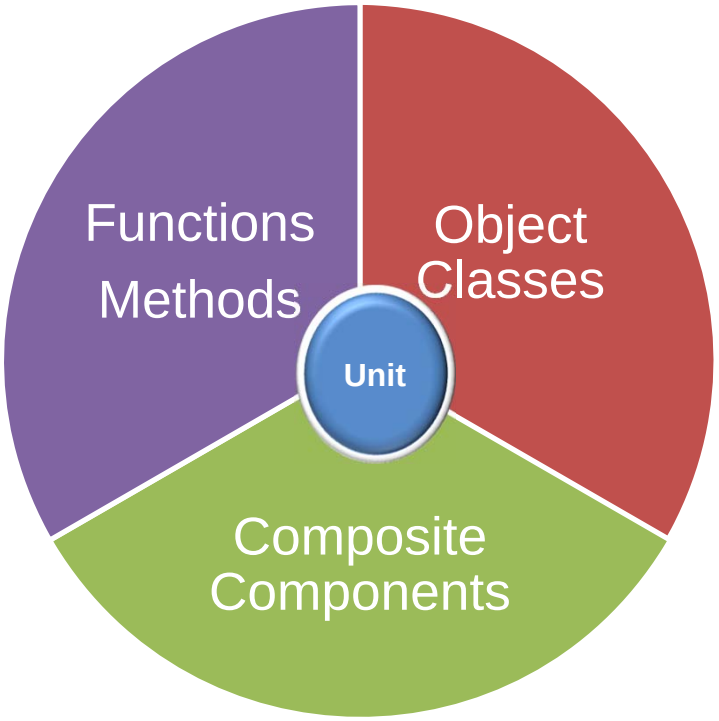
- Development testing includes all testing activities that are carried out by the team developing the system.
  - Unit testing, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
  - Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
  - System testing, where 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.

Development testing

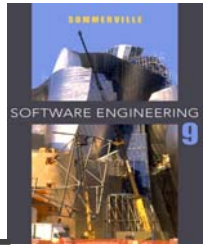


Unit testing

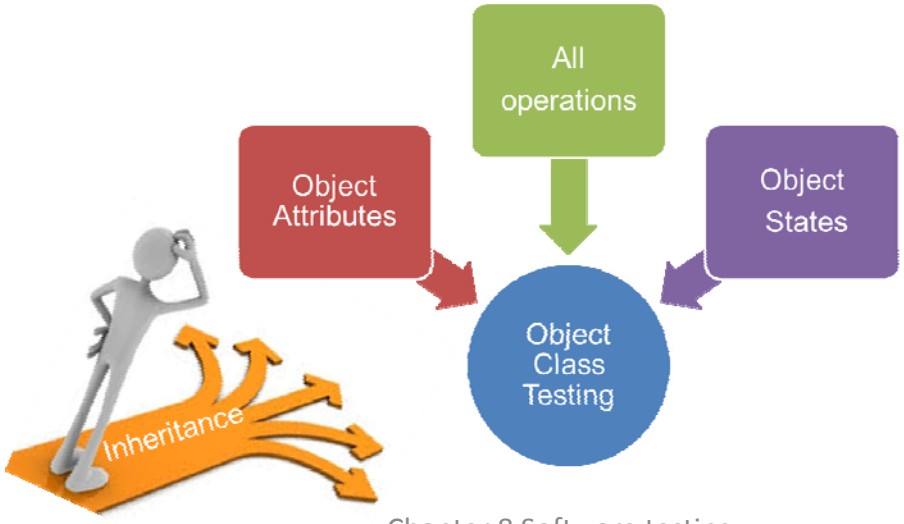
- Unit testing is the process of testing individual components in isolation.
- It is a defect testing process.



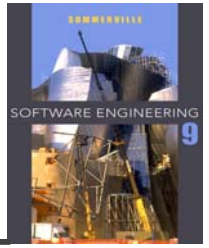
Object class testing



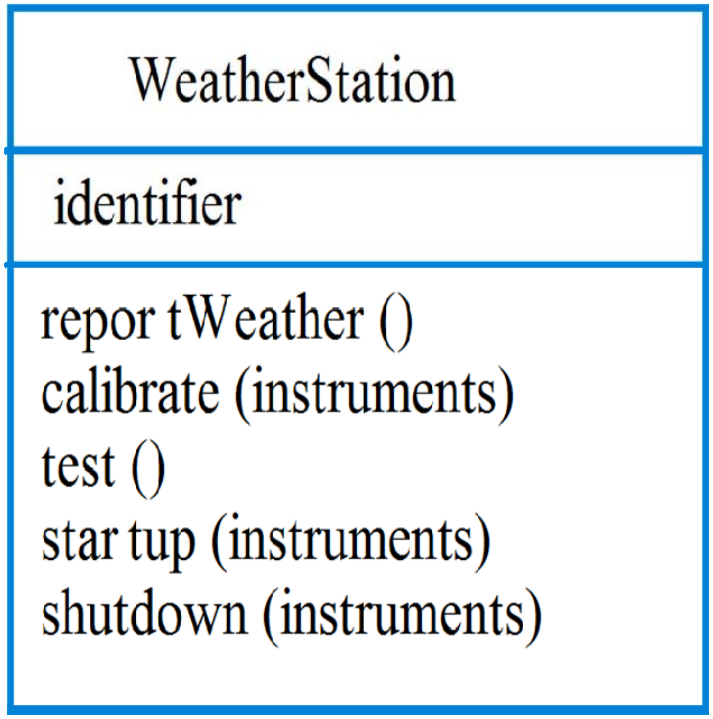
- Complete test coverage of a class involves
  - Testing all operations associated with an object
  - Setting and interrogating all object attributes
  - Exercising the object in all possible states.
- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.



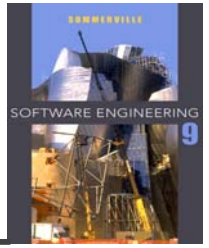
Weather station testing



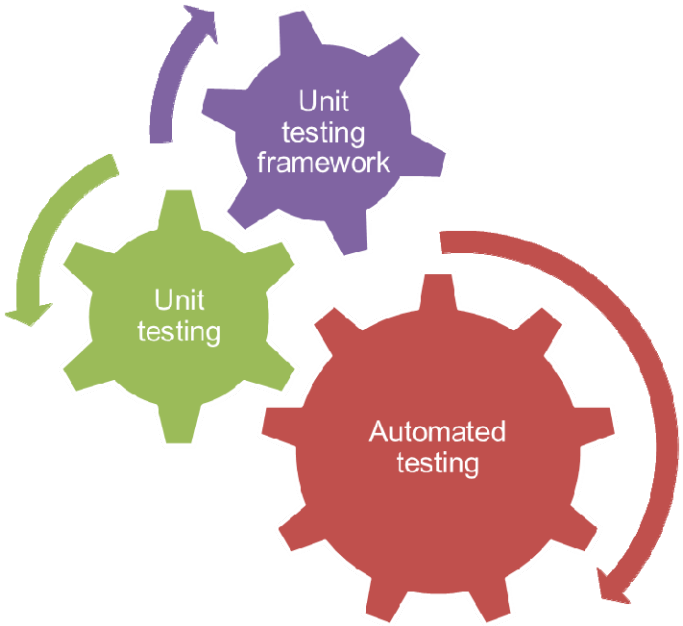
- Test Cases
  - reportWeather
  - Calibrate
  - test
  - startup
  - shutdown
- State model
  - sequences of state transitions
  - event sequences
- For example
  - Shutdown -> Waiting -> Shutdown
  - Waiting -> Calibrating -> Testing -> Transmitting -> Waiting



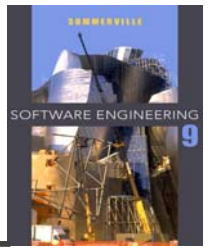
Automated testing



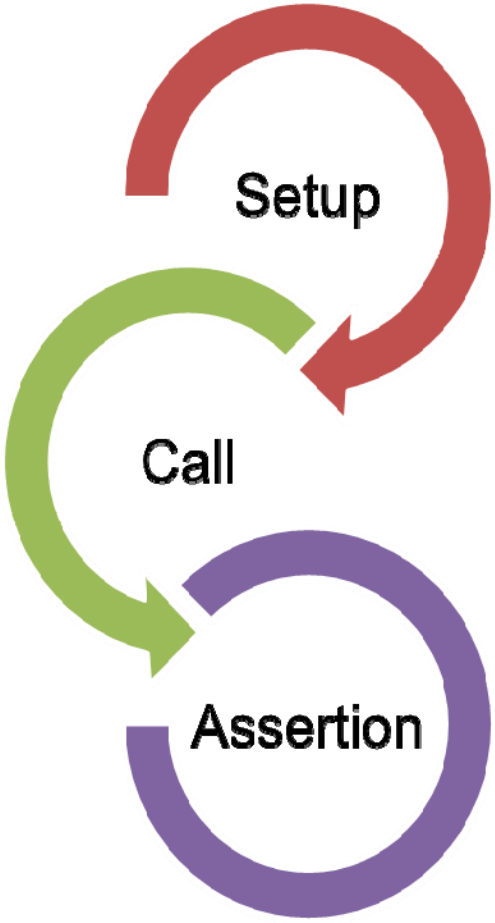
- Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.
- Unit testing frameworks provide generic test classes that you extend to create specific test cases. They can then run all of the tests that you have implemented and report, often through some GUI, on the success of otherwise of the tests.



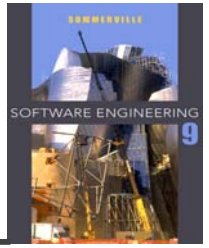
Automated test components



- A setup part, where you initialize the system with the test case, namely the inputs and expected outputs.
- A call part, where you call the object or method to be tested.
- An assertion part where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful if false, then it has failed.



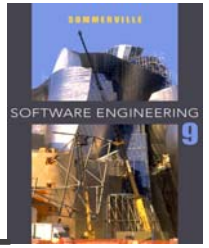
Unit test effectiveness



- The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.
- Existing defects in the component should be revealed by test cases.
- Two types of unit test case:
  - Normal Operations: Unit test case should reflect normal operation of a program as expected.
  - Abnormal Operations: Unit test case should use abnormal inputs to check that these are properly processed and do not crash the component.

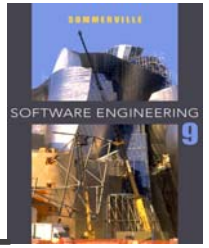
Testing strategies

Testing strategies



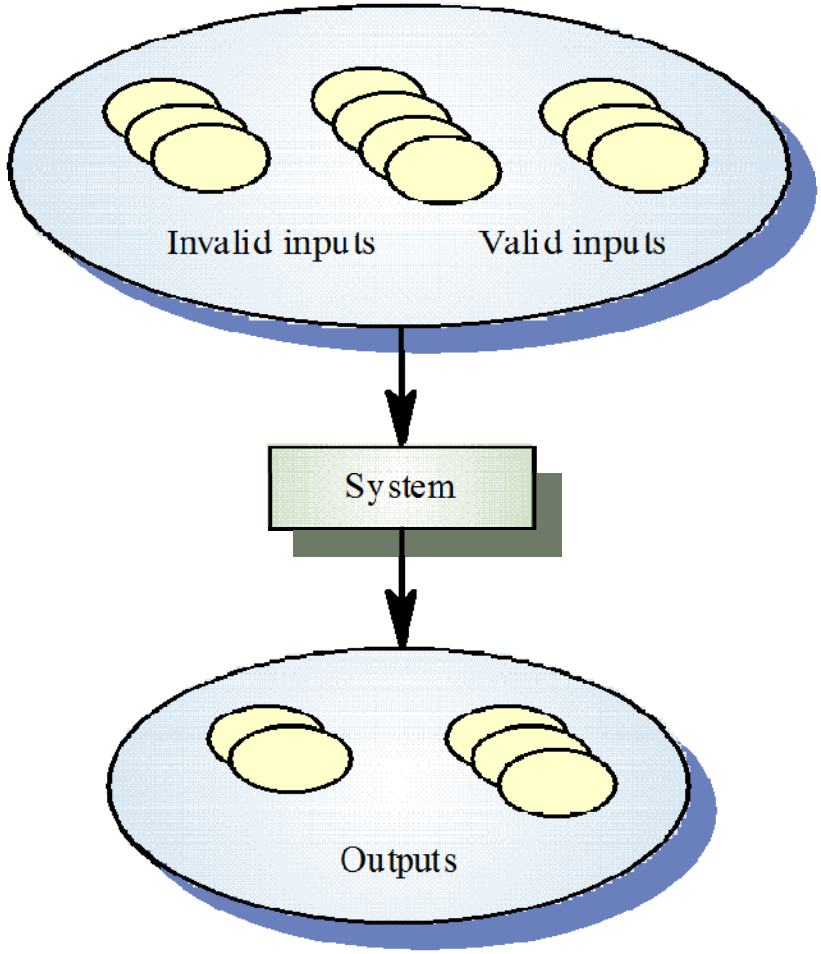
- Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- Guideline-based testing, where you use testing guidelines to choose test cases.
  - These guidelines reflect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

Partition testing

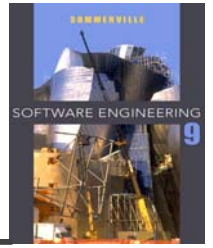


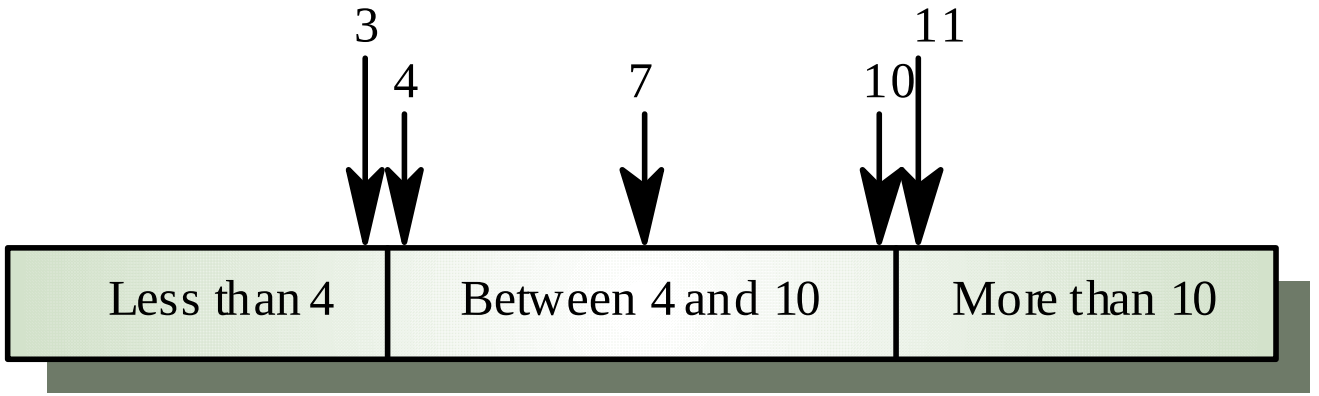
- Input data and output results often fall into different classes where all members of a class are related.
- Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.

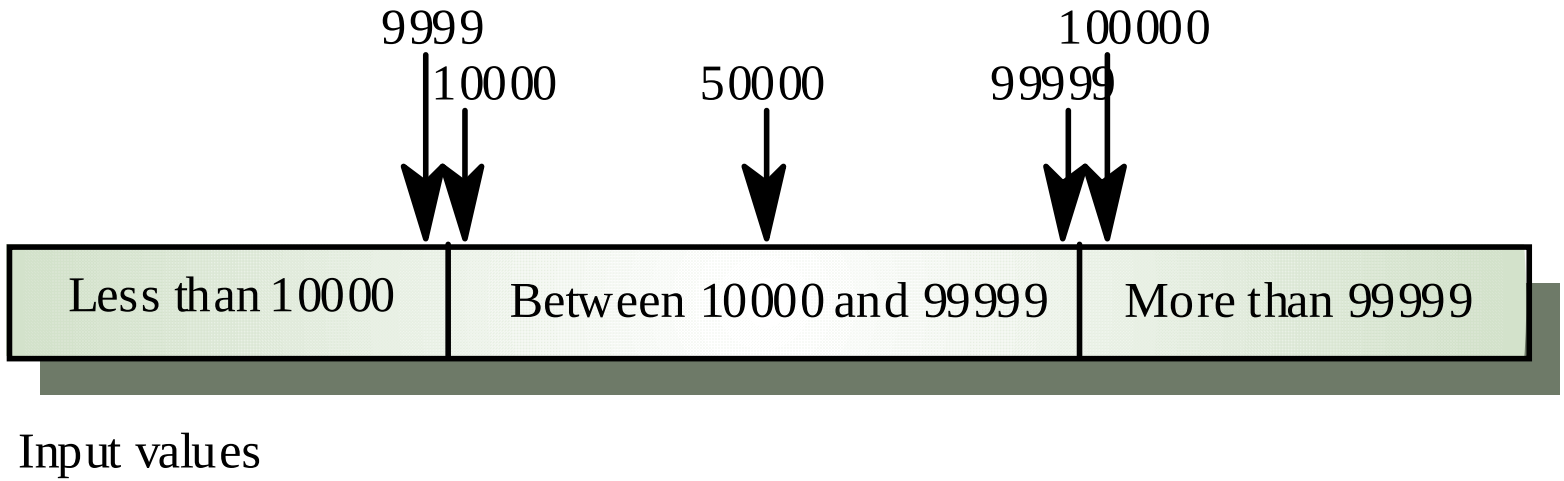
Equivalence partitioning



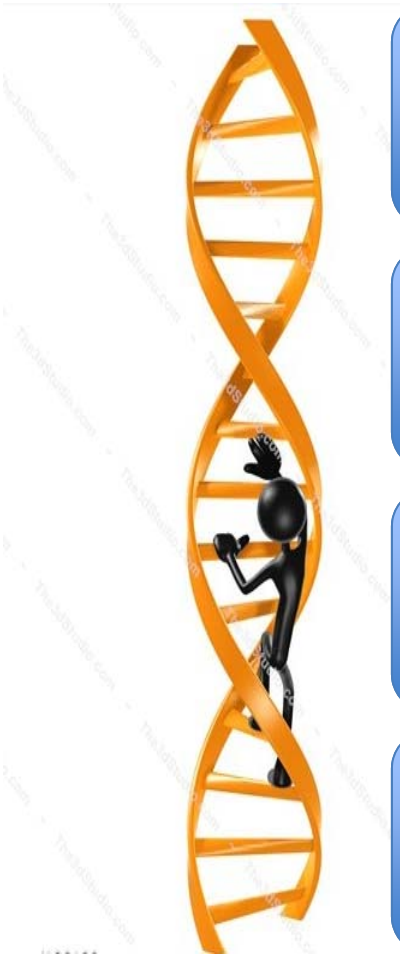
Equivalence partitions







Testing guidelines (sequences)



- Test software with sequences having a single value.
- Use sequences of different sizes in different tests.
- Tests to access the first, middle and last elements.
- Test with sequences of zero length.

General testing guidelines

Repeat the same input numerous times

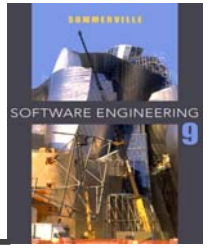
Choose inputs that force the system to generate all error

Design inputs that cause input buffers to overflow

Force invalid outputs to be generated

Force computation results to be too large or too small

Key points

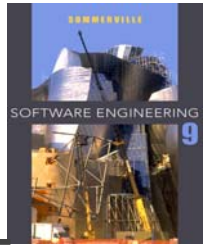


- Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.
- Development testing is the responsibility of the software development team. A separate team should be responsible for testing a system before it is released to customers.
- Development testing includes unit testing in which we test individual objects and methods, component testing in which we test related groups of objects and system testing in which we test partial or complete systems.

– Software Testing

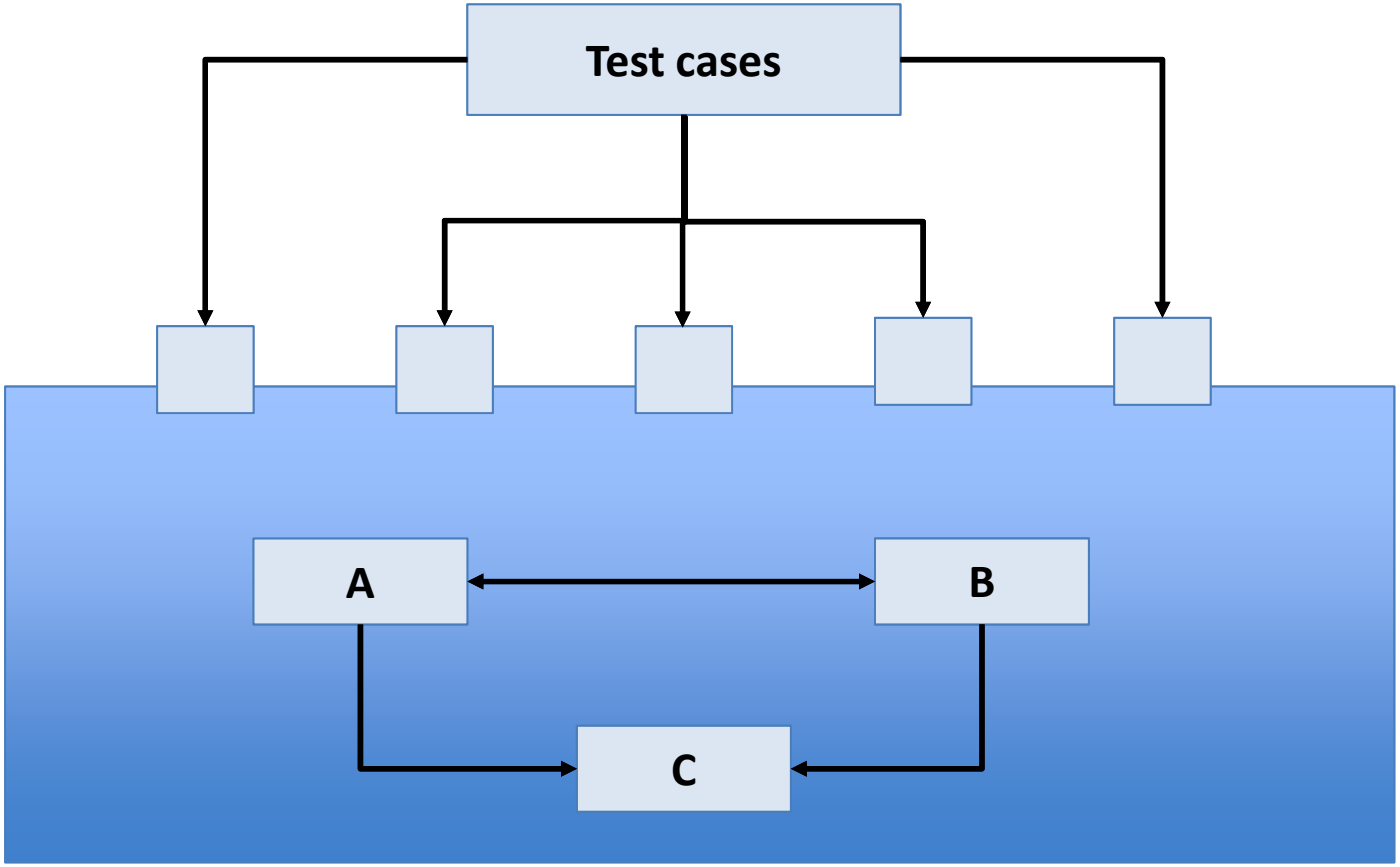
Lecture 2

Component testing

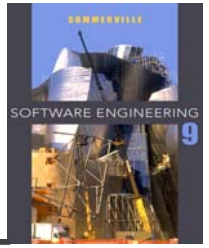


- Software component
  - Composite components of several interacting objects.
- Access the functionality
  - Through defined component interface.
- Testing composite components
  - Focus on showing that the component interface behaves according to its specification.

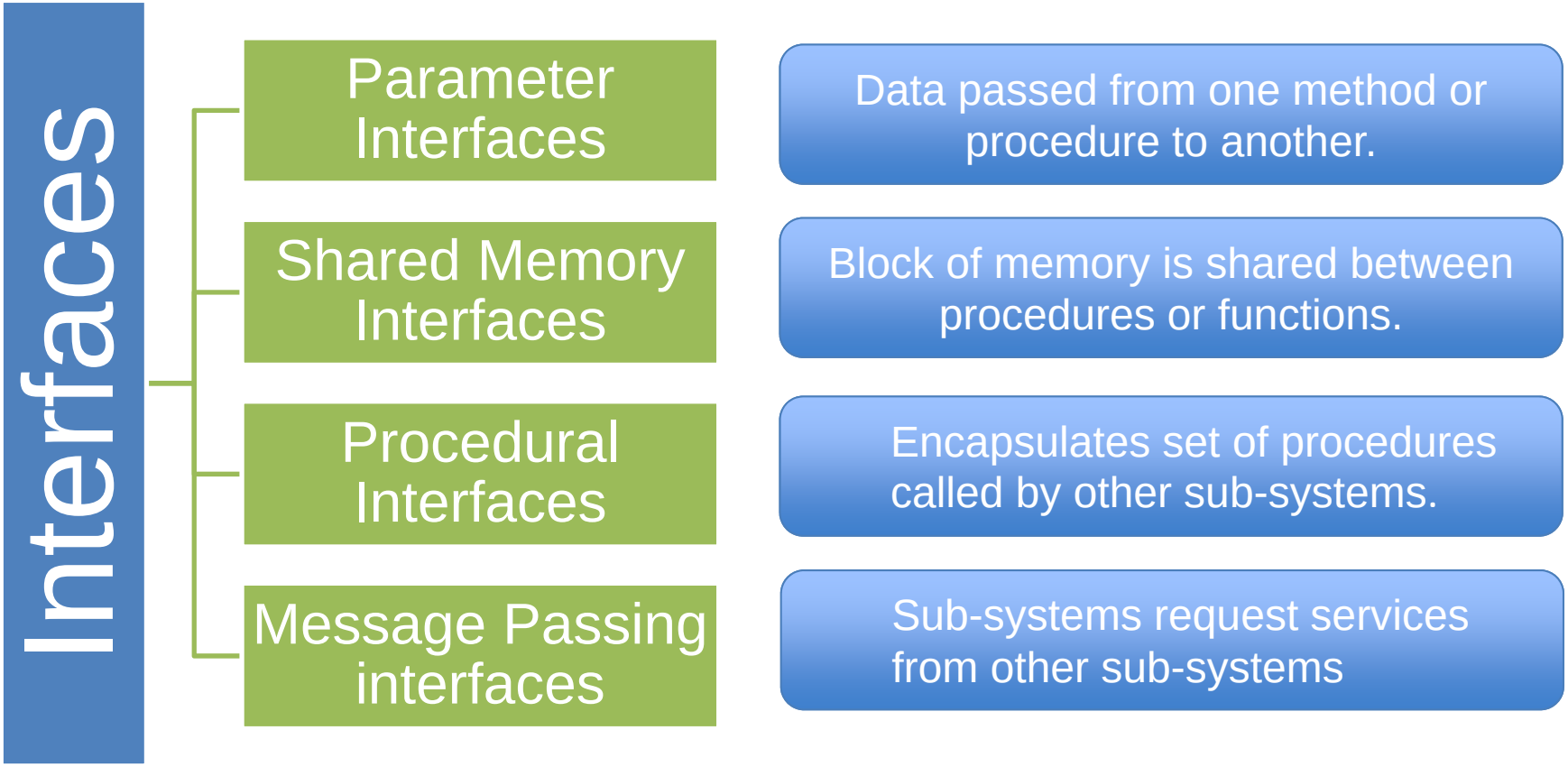
Interface testing



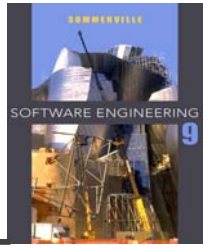
Interface testing



- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.



Interface errors



- A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.

Interface Misuse

- A calling component embeds assumptions about the behaviour of the called component which are incorrect.

Interface Misunderstanding

- The called and the calling component operate at different speeds and out-of-date information is accessed.

Timing Errors

Interface testing guidelines

Design tests with extreme ends parameters for called procedure.

Always test pointer parameters with null pointers.

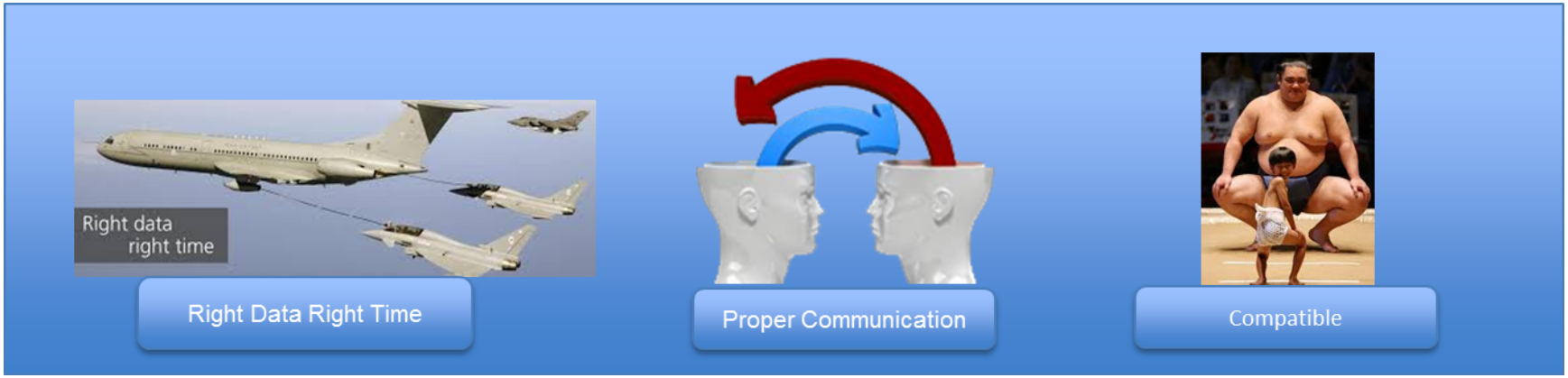
Design tests which cause the component to fail.

Use stress testing in message passing systems.

In shared memory systems, vary the activation order of components.

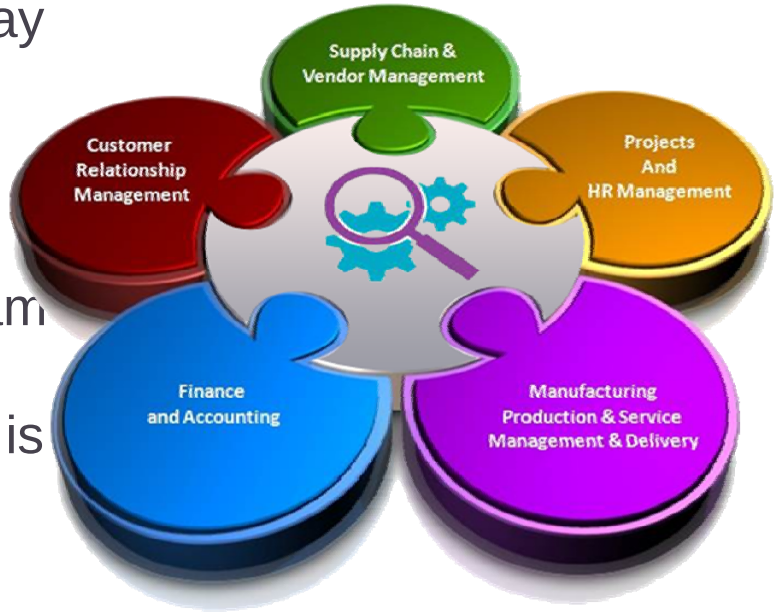
System testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing 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.

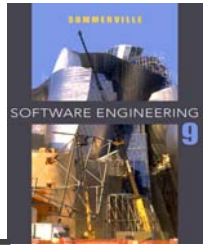


System and component testing

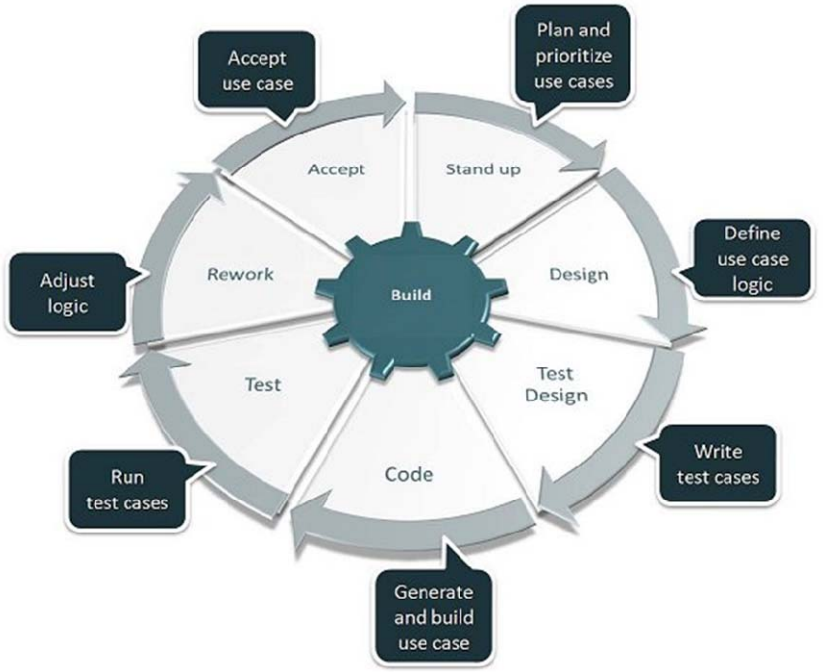
- During system testing, reusable components that have been separately developed and off-the-shelf systems may be integrated with newly developed components. The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage. System testing 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

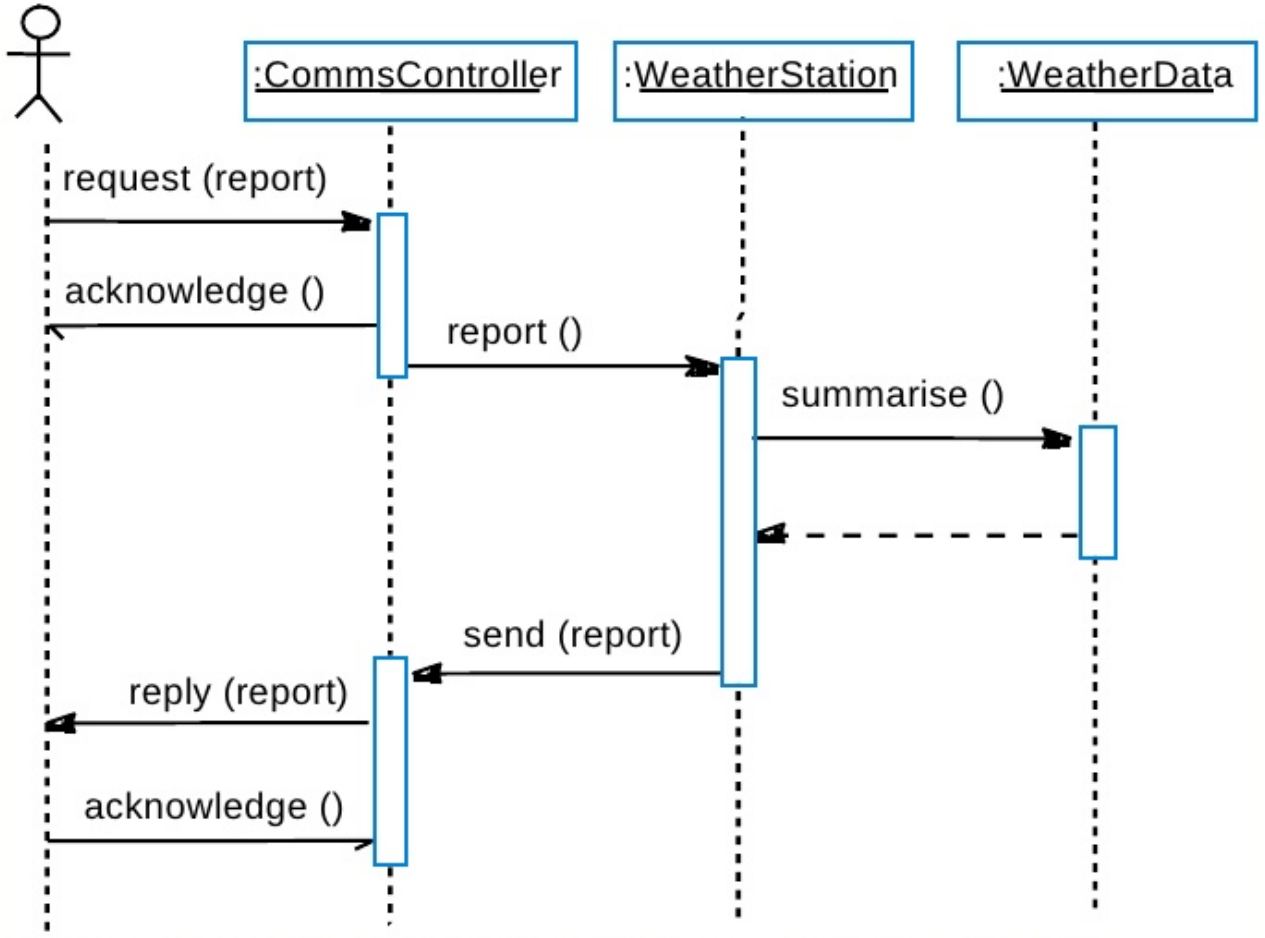


- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and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

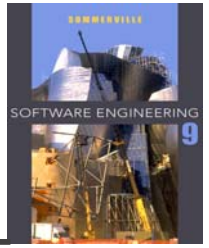


http://www.smartusecase.com/Default.aspx?Page=SmartUseCaseTesting&NS=&AspxAutoDetectCookieSupport=1

Collect weather data sequence chart



Testing policies

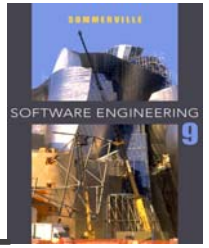


- Exhaustive system testing is impossible so testing policies which define the required system test coverage may be developed.
- Examples of testing policies:
  - All system functions that are accessed through menus should be tested.
  - Combinations of functions (e.g. text formatting) that are accessed through the same menu must be tested.
  - Where user input is provided, all functions must be tested with both correct and incorrect input.

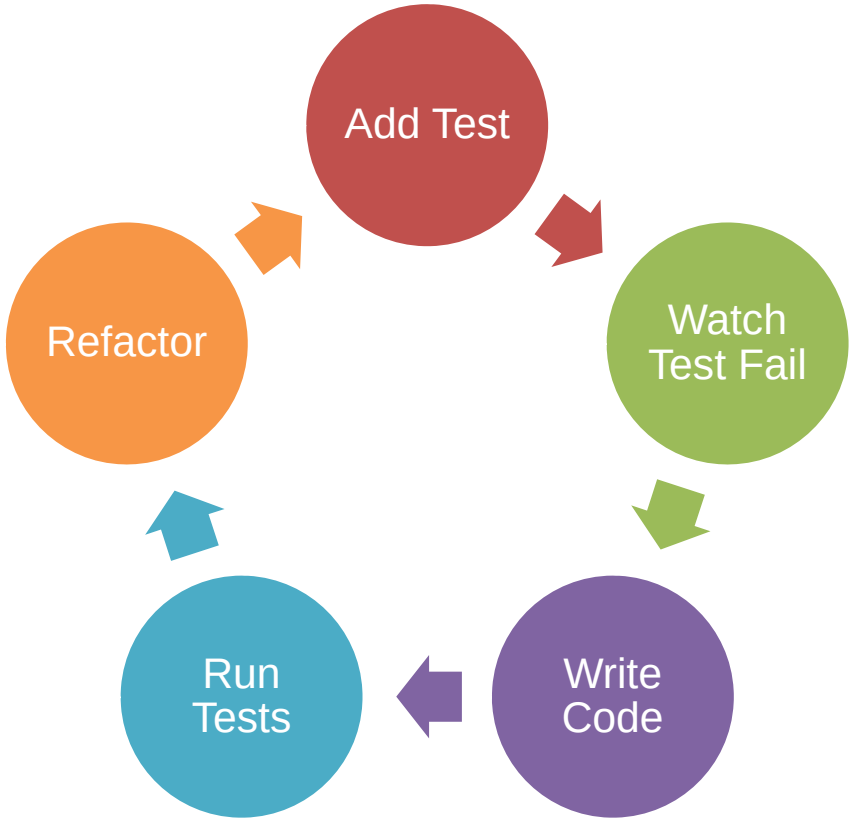
Exhaustive system testing is impossible

Testing policies for complete test system coverage

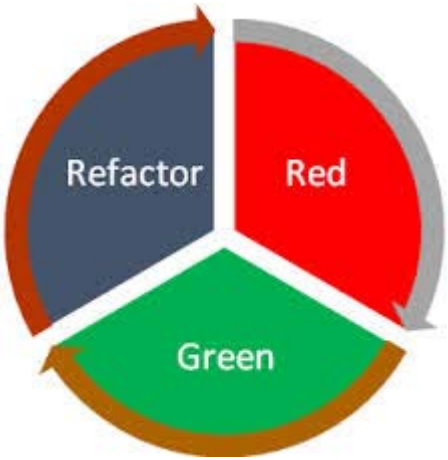
Test-driven development

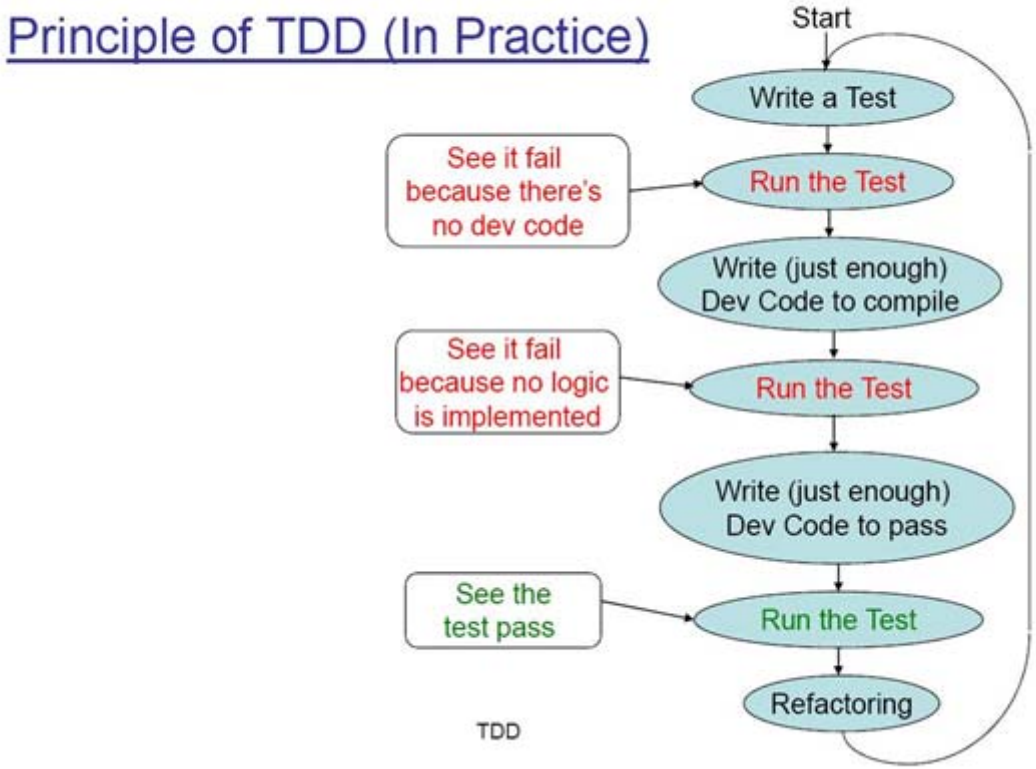


- Test Driven Development
  - Inter-leave testing & code development
- Test before Code
  - Passing the test is the critical driver of development
- Incremental Process
  - Passes test
  - Increment in code
- TDD part of Agile Method
  - Extreme Programming

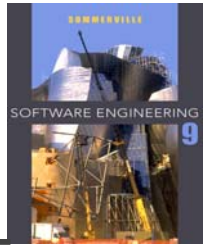


Test-driven development



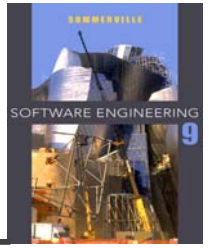


TDD process activities

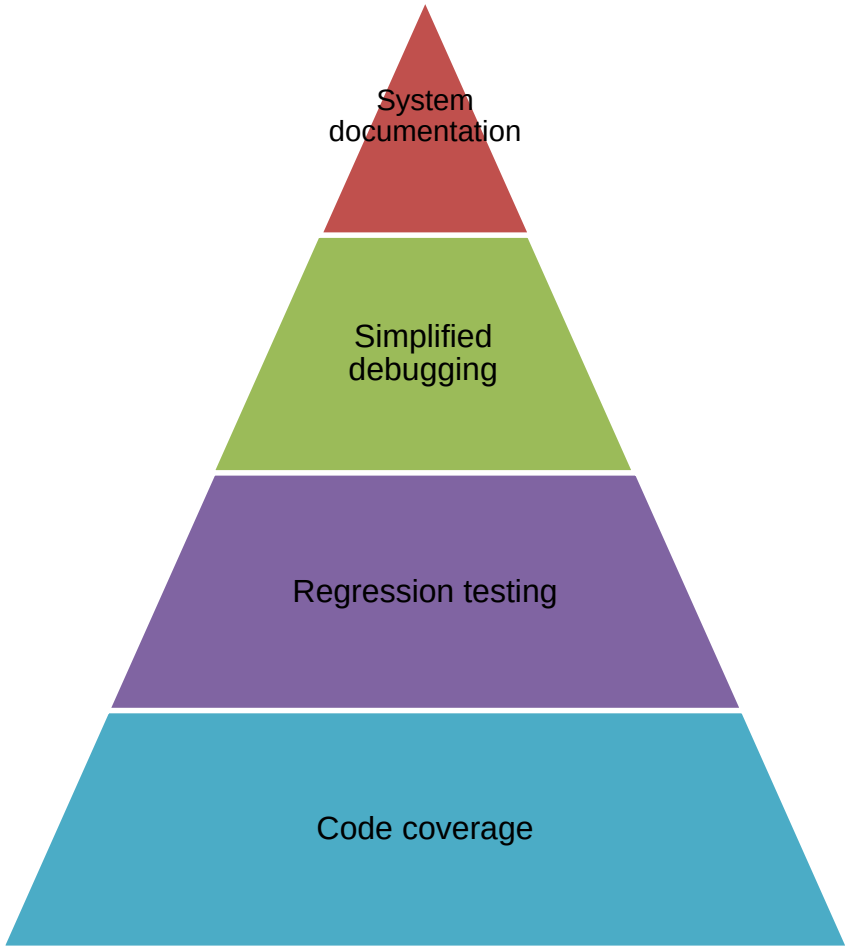


- Start by identifying the increment of functionality that is required. This should normally be small and implementable in a few lines of code.
- Write a test for this functionality and implement this as an automated test.
- Run the test, along with all other tests that have been implemented. Initially, 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.

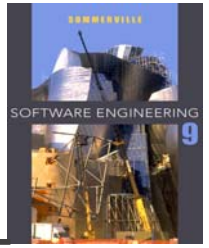
Benefits of test-driven development



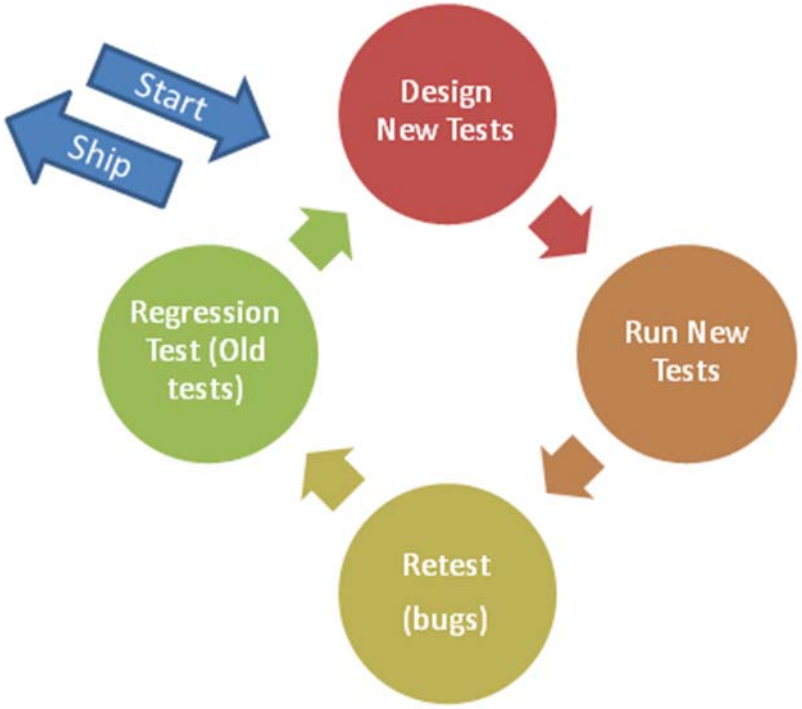
- Code coverage
  - Every code segment has at least one associated test.
- Regression testing
  - A regression test suite is developed incrementally.
- Simplified debugging
  - When a test fails, the newly written code needs to be checked and modified.
- System documentation
  - The tests themselves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



Regression testing

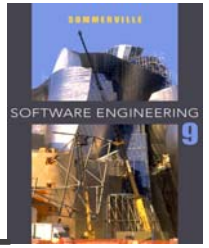


- Regression testing is testing the system to check that changes have not 'broken' previously working code.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.
- Tests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



http://www.softwaretestingclub.com/profiles/blogs/regression-testing-redefined

Release testing

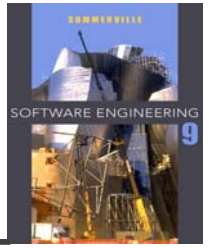


- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
  - Release testing, therefore, has to show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

Release testing

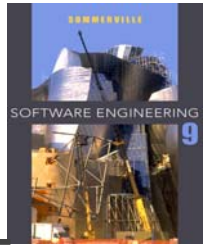
www.SoftwareTestingSoftware.com

Release testing and system testing

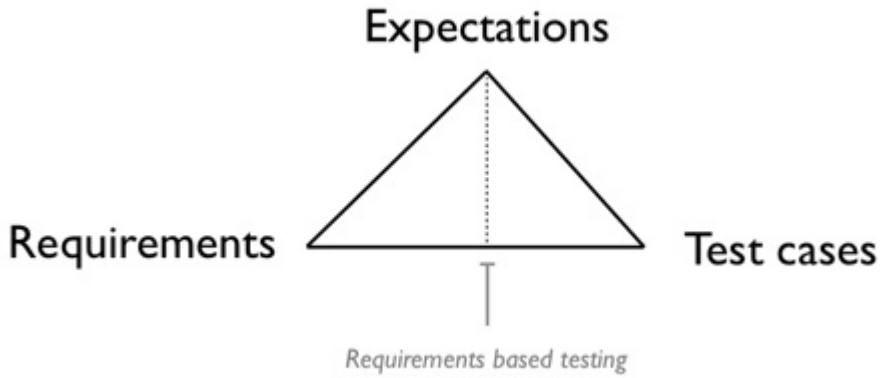


- Release testing is a form of system testing.
- Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing). The 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

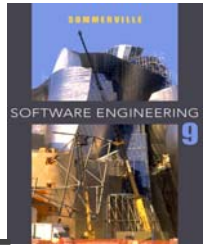


- Requirements-based testing involves examining each requirement and developing a test or tests for it.



http://bettersoftwareprojects.com/articles/debug-your-requirements/

MHC-PMS case study



- MHC-PMS 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

Requirement Test	Action	Result
Patient record with no known allergies	Prescribe medication for allergies	Warning message should not be issued by the system
Patient record with a known allergy	Prescribe the medication to that the patient is allergic to	Warning message should be issued by the system
Patient record in which allergies to two or more drugs are recorded	Prescribe both of these drugs separately	Correct warning message should be issued by the system for each drug
Patient record with two known allergic drug	Prescribe two drugs that the patient is allergic to	Two warnings should be issued correctly by system
Overrule the Warning	Prescribe a drug that issues a warning	The system should require the user to provide information explaining why the warning was overruled

Features tested by scenario

Authentication by logging

Downloading and uploading patient records

Home visit scheduling.

Encryption and decryption of patient records

Record retrieval and modification.

Links with the drugs database & side-effect information

System for call prompting.

A usage scenario for the MHC-PMS

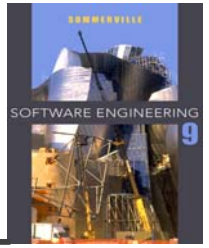
Kate is a nurse who specializes in mental health care. One of her responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side -effects.

On a day for home visits, Kate logs into the MHC-PMS and uses it to print her schedule of home visits for that day, along with summary information about the patients to be visited. She requests that the records for these patients be downloaded to her laptop. She is prompted for her key phrase to encrypt the records on the laptop.

One of the patients that she visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side -effect of keeping him awake at night. Kate looks up Jim's record and is prompted for her key phrase to decrypt the record. She checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so she notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. He agrees so Kate enters a prompt to call him when she gets back to the clinic to make an appointment with a physician. She ends the consultation and the system re-encrypts Jim's record.

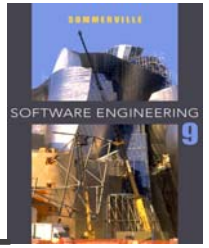
After, finishing her consultations, Kate returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for Kate of those patients who she has to contact for follow-up information and make clinic appointments.

Performance testing

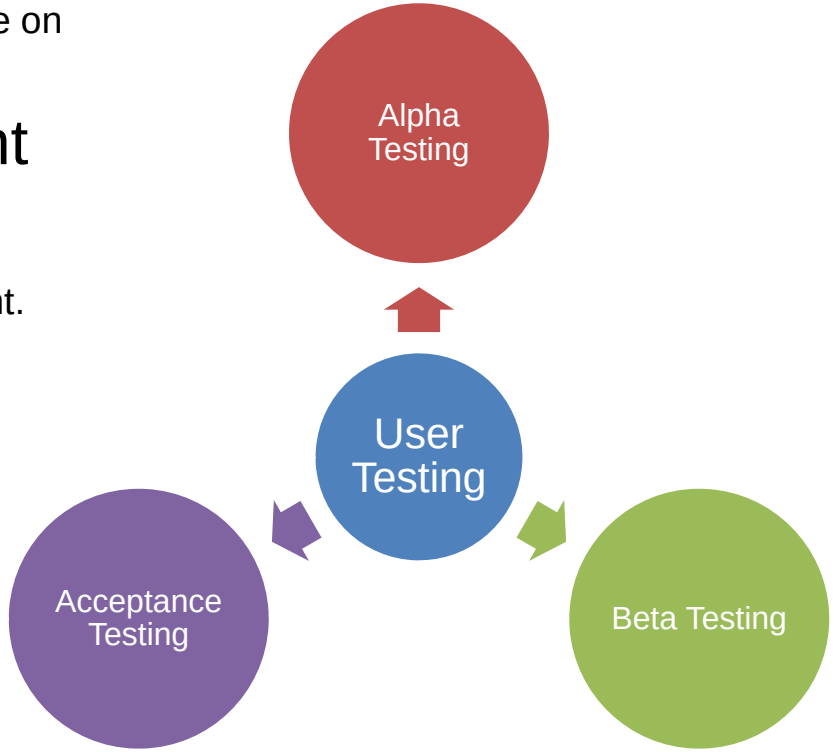


- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance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

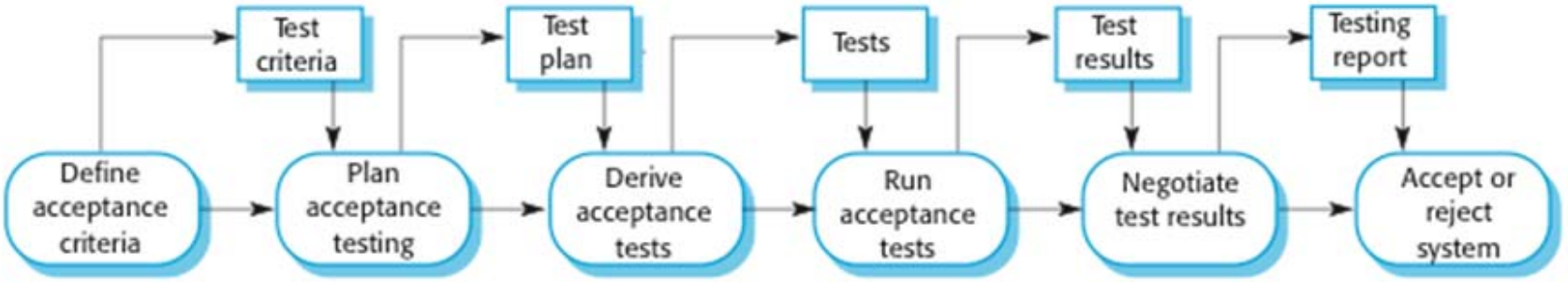
User testing & Its Types



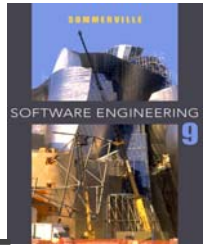
- User testing
  - Users or customers provide input and advice on system testing.
- Influence of Real environment
  - Major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.
- Alpha Testing
  - User work with development team
- Beta Testing
  - After release, user perform testing
- Acceptance Testing
  - Customers test system for deployment



The acceptance testing process

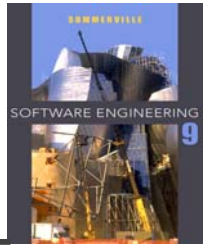


Stages in the acceptance testing process



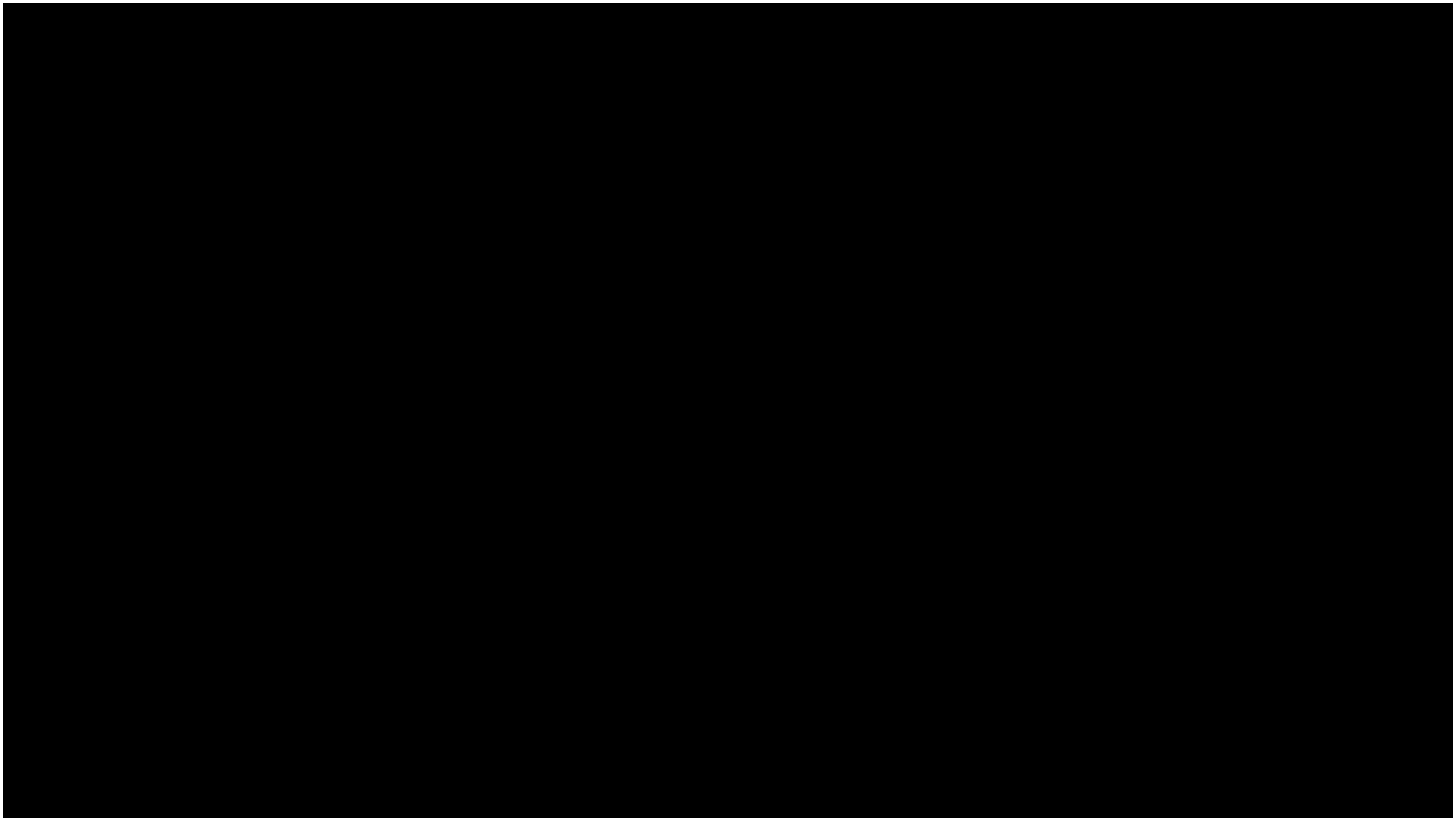
- Define acceptance criteria
- Plan acceptance testing
- Derive acceptance tests
- Run acceptance tests
- Negotiate test results
- Reject/accept system

Agile methods and acceptance testing

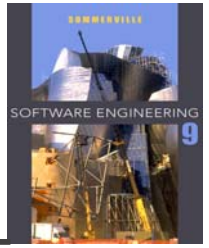


- In agile methods, the user/customer is part of the development team and is responsible for making decisions on the acceptability of the system.
- Tests are defined by the user/customer and are integrated with other tests in that they are run automatically when changes are made.
- There is no separate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

How Google Test



Key points



- 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.
- 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.