Testing Through the Life Cycle

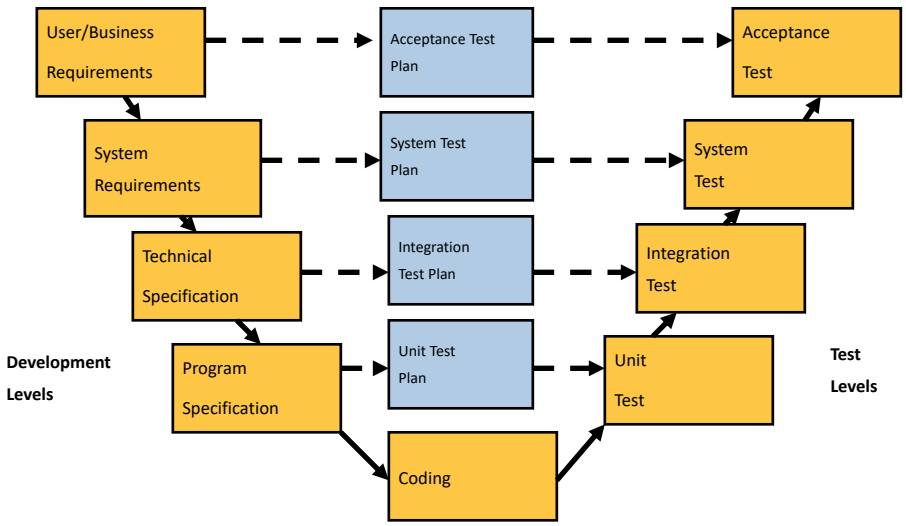
Outline

- Software Development models
- Test levels
- Test types
- Maintenance testing

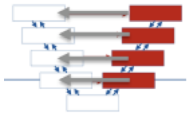
Principles for Good Testing

- Test does not come at the end, but it is integrated in the model!
- Only test execution is at the end, everything else as early as possible.
- The earlier one develops test cases, the earlier one can find bugs or possible bugs.

V-Model

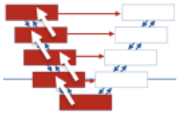


Validation



- In each stage of testing is to prove whether the development results meet the requirements which are specified or relevant for the respective level .
- To check the development results against the original requirements is called validation
- While validating the testers evaluate whether a (partial) product can actually solve a fixed (specified) task and therefore is suitable respectively useful for its purpose.
- Examines whether the product is useful in the context of the intended product use.

Verification



- Besides validating testing the V-model request so-called verification testing.
- Verification, unlike validation, is based on a single development phase and has to prove the correctness and completeness of a phase results relative to its direct specification (phase input documents).
- Examines whether the specifications have been implemented correctly, regardless of the intended purpose or use of the product.

V-Model

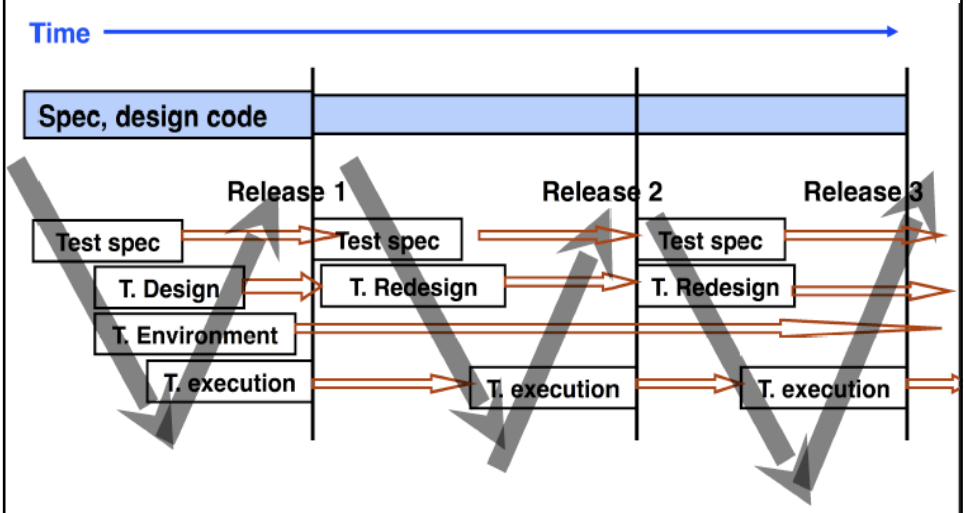
The benefits of the V Model include:

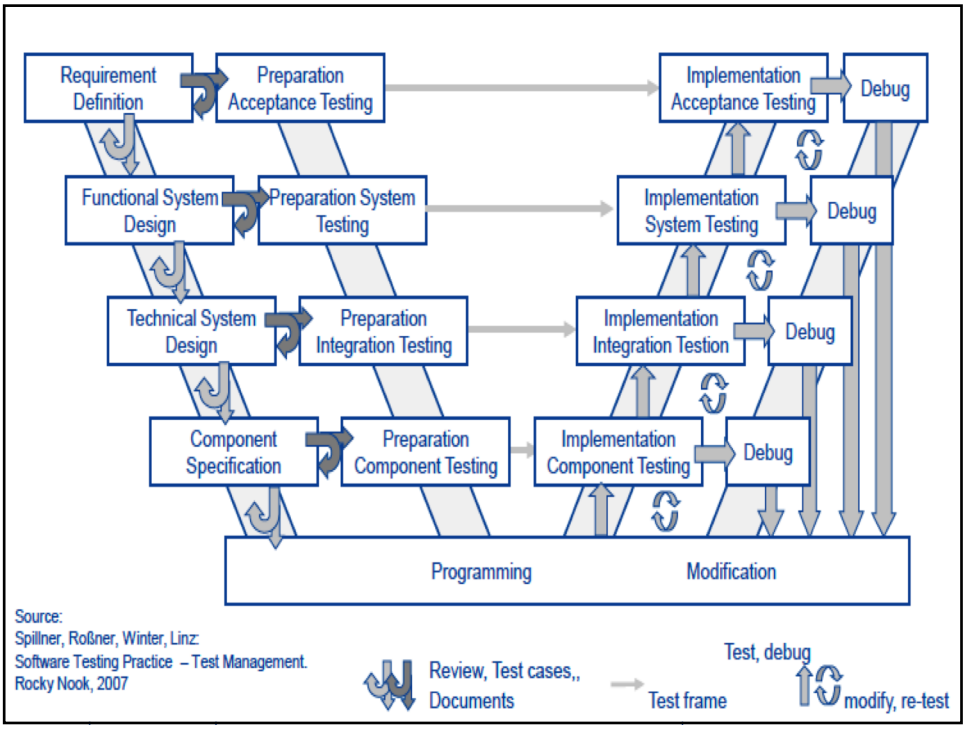
- The testing phases are given the same level of management attention and commitment as the corresponding development phases
- The outputs from the development phases are reviewed by the testing team to ensure their testability
- Verification and validation (and early test design) can be carried out during the development of the software work products
- The early planning and preliminary design of tests provides additional review comments on the outputs from the development phase

How to Apply the V-Model

- Run a V for every increment / release.
- Run a V on every prototype.
- Make test checklists as early as possible.
- Make test data during coding.
- Keep the test material, update and automate it.
  - Update it for every release as you learn more about defects.
- Regression test for every release!
  (Run a smoke test for every Build)

The Real Way of Working with Test





Principles for Good Testing in All Life Cycle Models

- Every development activity has its corresponding test activity.
- Every test level has a specific objective.
- Test preparation shall start at the same time as the corresponding development activity → feedback, thinking, improvement / adaptation.
- Testers can review drafts for development documents, at the same time as test design.
- The number of test levels depends on system complexity.
  - Especially integration test is often partitioned into several levels. Test levels can also be combined.

Test Levels

- Component Testing
- Integration Testing

} Fault-directed testing

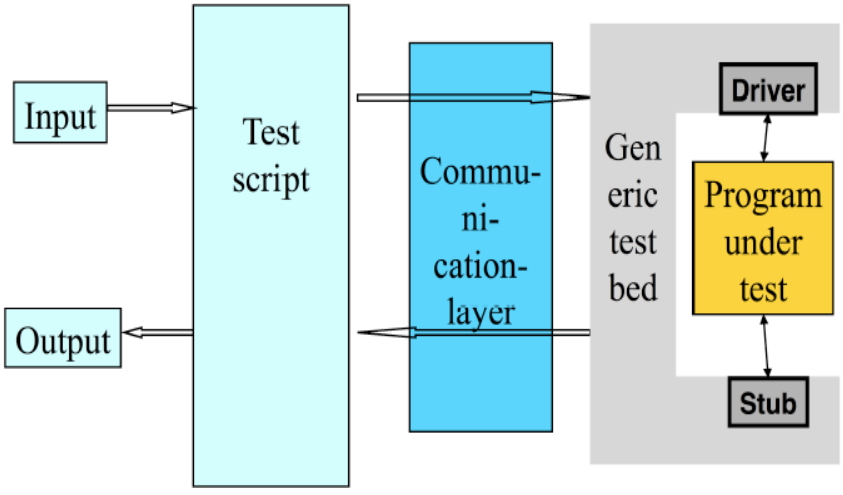
- System Testing
- Acceptance Testing

} Conformance-directed testing

Tasks for the Test Levels

- Component (Unit, Module) test:
  - Coding problems, problems in detailed design, algorithms.
- Integration test:
  - Problems in interfaces, working together, design, architecture.
- System test:
  - Problems in requirements, problems with system attributes.
- Acceptance test:
  - Deviations from the customer interpretation of requirements and needs. The system does not work on the customer platform and in the customer environment.

Design of Test Environment

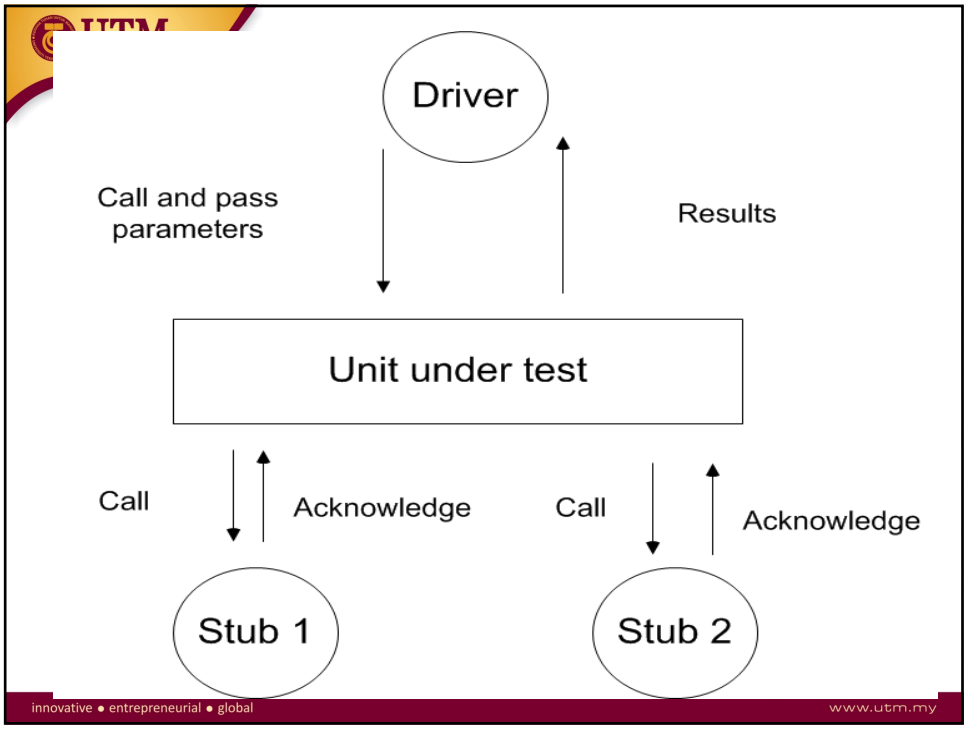


Test Environment

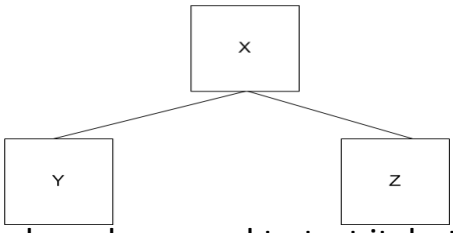
- Test environment contains drivers and stubs.
- Driver:
  - A piece of code that passes test cases to another piece of code.
  - is used to simulate a calling module and call the program unit being tested by passing input arguments.
  - The driver can be written in various ways, (e.g., prompt interactively for the input arguments or accept input arguments from a file).
  - Can be generated automatically by tools.

Test Environment

- Stubs:
  - A Stub is a dummy procedure, module or unit that stands in for an unfinished portion of a system.
  - Four basic types of Stubs for Top-Down Testing are:
    1 Display a trace message
    2 Display parameter value(s)
    3 Return a value from a table
    4 Return table value selected by parameter
  - More than one stub may be needed, depending on the number of programs the unit calls.
  - Can be generated automatically by a tool.
  - Should be kept for regression test.



Example



- X module is ready and we need to test it, but it calls functions from y and z (which are not yet ready).
  - To test at such a module, we write a small dummy piece a code which Simulates Y and Z and which will return values for X.
  - This piece of dummy code is called Stub in a Top down Integration Testing.

- Y and Z modules ready and x module is not ready, and we need to test Y and Z modules which accepts values from X,
  - To get the values from X, a small piece of dummy code for X which returns values for Y and Z is needed.
  - So this piece of code is called Driver in Bottom up Integration testing.

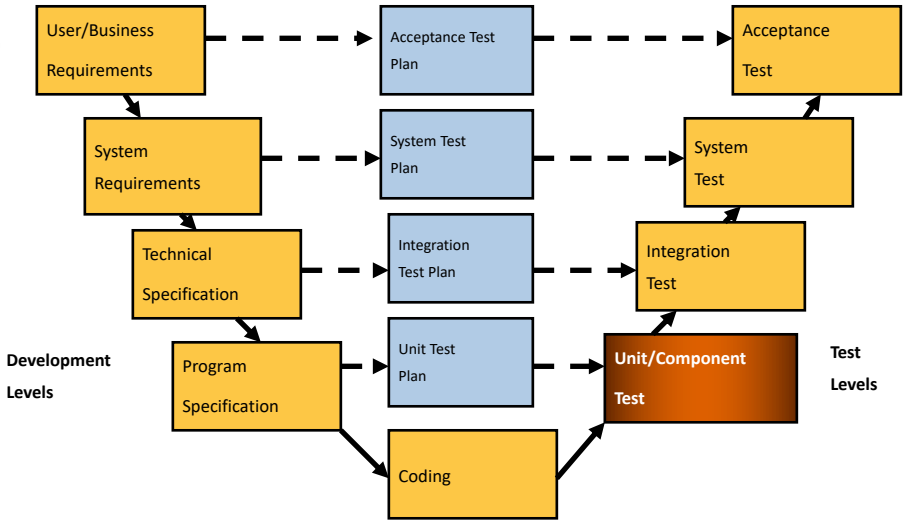
Stub Example

```
public int generateRandInt()
{
    return 1;
}
```

Driver Example

```
public class RandIntTest
{
    public static void main(String[] args)
    {
        RandInt myRand = new RandInt();
        System.out.println("My first rand int is
:" + myRand.generateRandInt());
    }
}
```


Component Testing



Component Testing

Definition

- **Component** – *A minimal software item that can be tested in isolation.*
- **Component Testing** – *The testing of individual software components*.
- Sometimes known as Unit Testing, Model Testing or Program Testing
- Component can be tested in isolation – stubs/drivers may be employed
- Test cases derived from component specification (module/program spec)
- Functional and Non-Functional testing
- Usually performed by the developer, with debugging tool
- Quick and informal defect fixing

Component Testing

Definition

- Test-First/Test-Driven approach – create the tests to drive the design and code construction!
- Instead of creating a design to tell you how to structure your code, you create a test that defines how a small part of the system should function.
- Three steps:
  1. Design test that defines how you think a small part of the software should behave (Incremental development).
  2. Make the test run as easily and quickly as you can. Don't be concerned about the design of code, just get it to work!
  3. Clean up the code. Now that the code is working correctly, take a step back and re-factor to remove any duplication or any other problems that were introduced to get the test to run.

Russell Gold, Thomas Hammell and Tom Snyder - 2005

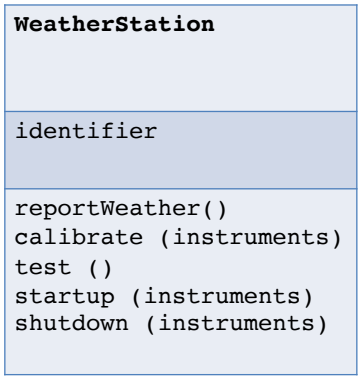
Component Testing

- Exercising the smallest individually executable code units
- It is a defect testing process.
- Component or unit testing is the process of testing
- individual components in isolation.
- Objectives
  – Finding faults
  – Assure correct functional behaviour of units
- Usually performed by programmers

Component Testing

- Components 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.
- 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.

An Example of Object Class Testing



- Need to define test cases for `reportWeather, calibrate, test,`
- `startup` and `shutdown`.
- Using a state model, identify sequences of state transitions to be tested and the event sequences to cause these transitions.
- For example:
  – Waiting -> Calibrating -> Testing -> Transmitting -> Waiting

Integration Testing

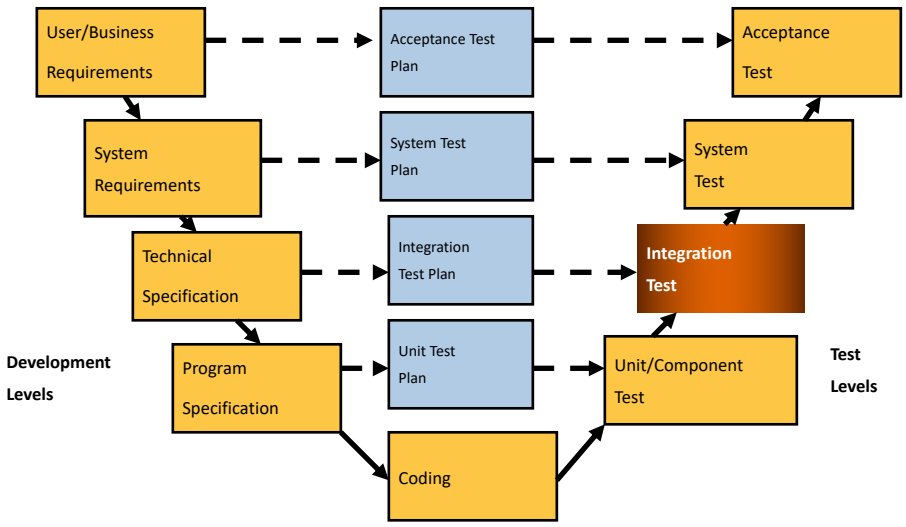
- Definition
- Component Integration Testing
- System Integration Testing

Integration Testing

Definition

- **Integration Testing** - *Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems*
- Components may be code modules, operating systems, hardware and even complete systems
- There are 2 levels of Integration Testing
  - Component Integration Testing
  - System Integration Testing

Component Integration Testing



Component Integration Testing

Definition

- **component integration testing** *Testing performed to expose defects in the interfaces and interaction between integrated components*
- Usually performed by the Developer, but could involve the test team usually formal (records of test design and execution are kept)
- all individual components should be integration tested prior to system testing

Component Integration Testing

Test Planning

- To consider - should the integration testing approach:
  - Start from top level components and work down?
  - Start from bottom level components and work up?
  - Use the big bang method?
  - Be based on functional groups?
  - Start on critical components first?
  - Be based on business sequencing? Maybe suit System Test needs.
- Knowledge of the system architecture is important
- The greater the scope of the integration approach the more difficult it is to isolate defects
- Non-Functional requirements testing may start here – e.g. early performance measures

System Testing

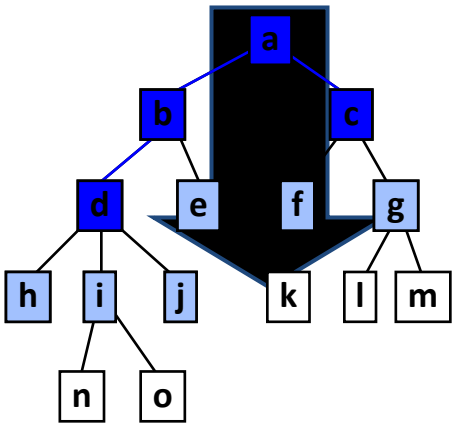
- Context
- Definition
- Functional Systems testing
- Non-Functional Systems Testing
- Good Practices for System Testing

Integration Strategies

- Depend on system architecture
- Depend on cost for test environments (drivers, stubs).

Top-Down Integration

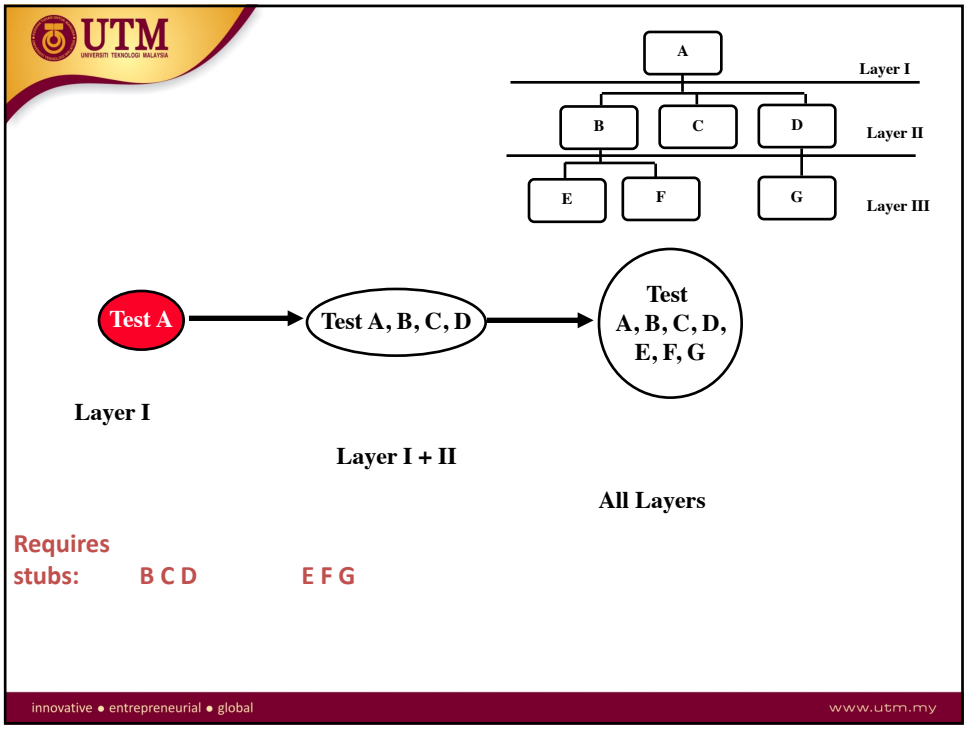
- Baselines:
  - baseline 0: component a
  - baseline 1: a + b
  - baseline 2: a + b + c
  - baseline 3: a + b + c + d
  - etc.
- Need to call to lower level components not yet integrated



Component Integration Testing

Top-down testing

- Test commences with the top module in the system and tests in layers descending through the dependency graph for the system.
- This may require successive layers of `stub' modules that replace modules lower in the dependency graph.



Component Integration Testing

Top-down testing

Pro's

- provides a limited working system early in the design process
- depth first integration demonstrates end-to-end functions early in the development process
- early detection of design errors through early implementation of the design structure
- early testing of major control or decision points
- this approach may allow an overlap with Component Testing

Con's

- stubs only provide limited simulations of lower level components and could influence spurious results
- breadth first means that higher levels of the system must be artificially forced to generate output for test observations

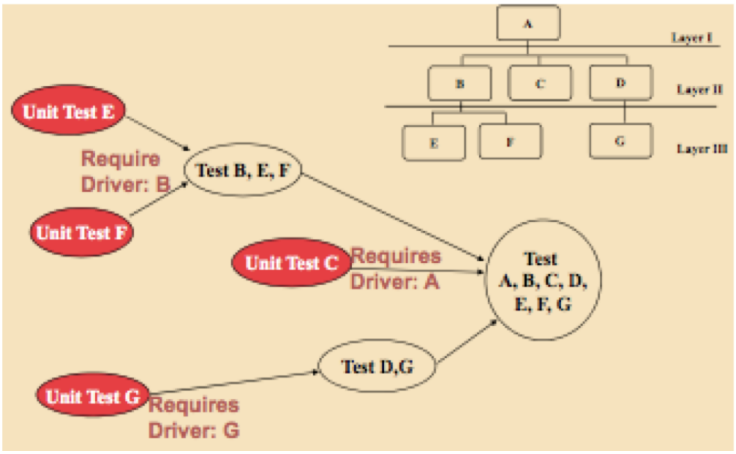
Component Integration Testing

Bottom-up testing

- Initiate testing with unit tests for the bottom modules in the dependency graph
- Candidates for inclusion in the next batch of tests depend on the dependency structure (a module can be included if all the modules it depends on have been tested (issue about potential circularity need to consider connected components).
- Prioritisation of modules for inclusion in the test sequence should include their `criticality' to the correct operation of the system.

Component Integration Testing

Bottom-up testing



Component Integration Testing

Bottom-up testing

Pro´s

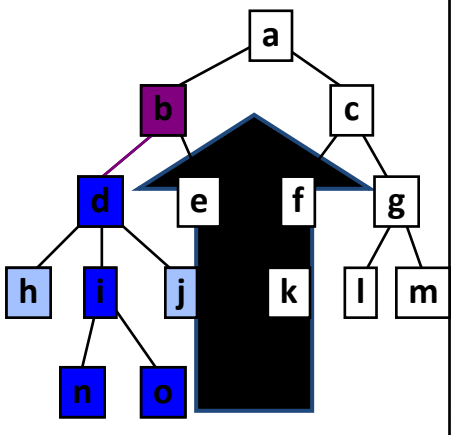
- using drivers instead of upper level modules to simulate the environment for lower level modules
- necessary for critical, low level system components
- testing can be observed on the components under test from an early stage

Con´s

- unavailability of a demonstrable system until late in the development process
- late detection of system structure errors

Bottom-up Integration

- Baselines:
  - baseline 0: component n
  - baseline 1: n + i
  - baseline 2: n + i + o
  - baseline 3: n + i + o + d
  - etc.
- Needs drivers to call the baseline configuration
- Also needs stubs for some baselines



Component Integration Testing

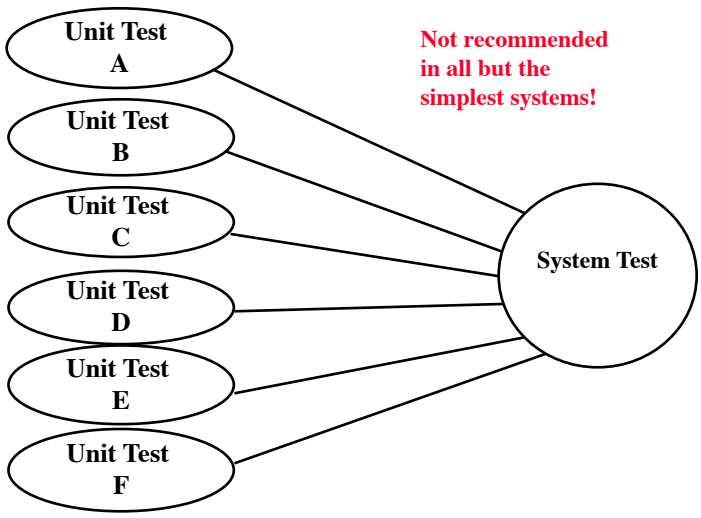
Big Bang Approach

In theory:

- if we have already tested components why not just combine them all at once? Wouldn't this save time?
- (based on false assumption of no faults)

In practice:

- takes longer to locate and fix faults
- re-testing after fixes more extensive
- end result? takes more time



Component Integration Testing

Suggested Integration Testing Methodology

The following testing techniques are appropriate for Integration Testing:

- *Functional Testing* using Black Box Testing techniques against the interfacing requirements for the component under test
- *Non-functional Testing* (where appropriate, for *performance* or *reliability testing* of the component interfaces, for example)

Steps in Integration Testing

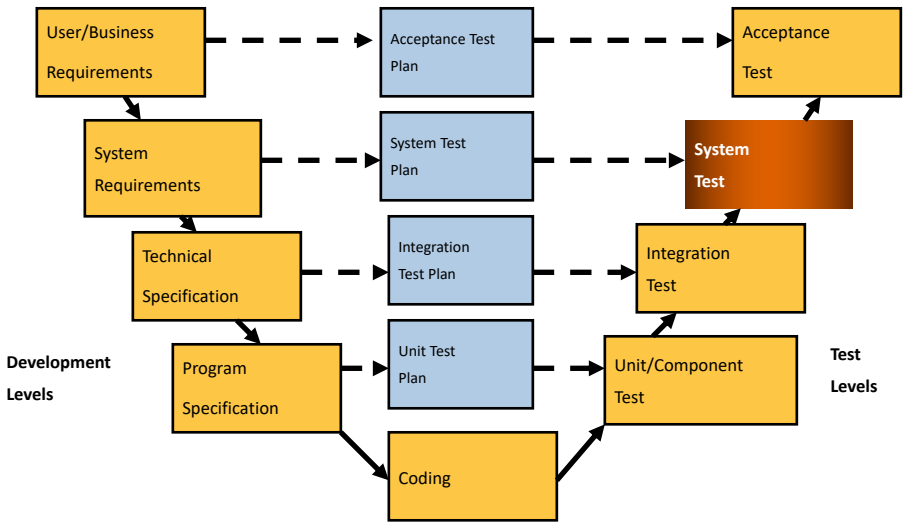
1. Based on the integration strategy, *select a component* to be tested. Unit test all the classes in the component.
2. Put selected component together; do any *preliminary fix-up* necessary to make the integration test operational (drivers, stubs)
3. Do *functional testing:* Define test cases that exercise all uses cases with the selected component
4. Do *structural testing:* Define test cases that exercise the selected component
5. Execute *performance tests*
6. *Keep records* of the test cases and testing activities.
7. Repeat steps 1 to 7 until the full system is tested.

The primary *goal of integration testing is to identify errors* in the (current) component configuration.

Tester Tasks with Integration

- Early feedback
  - Which interfaces make testing difficult?
  - Which components must be delivered first (to ease integration?)

System Testing



System Testing

Definition

- **System Testing -** *process of testing an integrated system to verify that it meets specified requirements*

- Concerned with the behaviour of the whole system, not with the workings of individual components.
- Carried out by the Test Team

System Testing

- Typical test objects:
  - System, user and operation manuals, system configuration and configuration data.
- Test basis:
  - System and SRS, functional specifications, risk analysis reports.

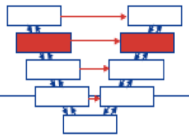
System Testing – What to include?

- Function test
- Function interaction, flow
- Non-functional attributes
  - Efficiency (stress, load, volume test)
  - Usability
  - Safety
  - Robustness
  - Portability
  - Maintainability

System Testing – Who?

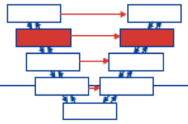
- Normally independent test team
- Large task to make test environment
- Customer viewpoint
- Problem:
  - Often incomplete / undocumented requirements!!!

System Testing – Test Harness (1)



- The test environment should correspond as much as possible to the final production environment.
- Instead of drivers and stubs, the actual hard- and software products used should be installed in the test environment (hardware equipment, system software, driver software, network, external systems etc.).

System Testing – Testing Objective (1)

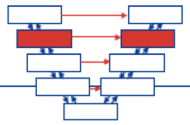


→ How well does the completed system fulfill the stated requirements?

- Two classes of requirements are considered separately:
  - Functional requirements
  - Non-functional requirements

- Functional requirements
  - Specify the behavior which the system or system parts have to perform.
  - They describe »what« the system(part) has to perform. Their implementation is required in order for the system to be suitable for use.
  - Characteristics of the quality attribute "functionality" according to [ISO 9126] are: suitability, accuracy, interoperability, security, compliance.

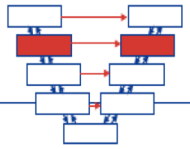
System Testing – Testing Objective (2)



→ How well does the completed system fulfill the stated requirements?

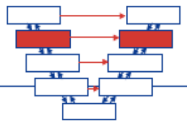
- Non-functional requirements
  - Describe »how well« certain attributes of the system(part) behavior should be performed.
  - Its implementation strongly influences how satisfied the customer or user will be with the product and how much they like using it.
  - Quality attributes according to [ISO 9126] are: reliability, efficiency, usability. Indirectly, maintainability and portability may also influnce the customer‘s satisfaction.
  - With some projects data is the main focus, e.g. with data conversion projects and with applications like Data Warehouses. Requirements relating to data quality must also be considered in the system test.

System Testing – Testing Strategy Functional Requirements (1)



- In the project phase the requirements were collected and recorded in a requirements specification document
- **Testing based on requirements**
  - The approved requirements specification document is used as test basis.
  - The system specification will also be verified by a review.
  - For each requirement (functional and non-functional) test cases are derived and recorded in the system specification.

System Testing – Testing Strategy
Functional Requirements (2)

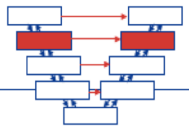


Business process-based testing

- Used if a software system has the purpose of automating or supporting the business process of a customer.
- A business process analysis (mostly created as part of the requirements analysis) shows which business processes are relevant, how often they are used and in which context they appear (persons, companies, external systems etc.)
- Subsequently, test scenarios (sequences of test cases) are set up on the basis of this analysis that represent typical business situations.
- The priority of the test scenarios depends on the amount and relevance of the particular business processes.

- Whereas testing based on the requirements focuses on individual system functions, a business process-based strategy focuses on a series of connected tests.

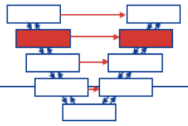
System Testing – Test Strategy
Non-functional Requirements (1)



- Non-functional requirements are just as important and eligible as functional requirements, as they determine important quality attributes which the system must posess.
- The following non-functional system properties should be taken into account (normally in the system testing):
  - **Load Testing**
    - Measurements of the system behavior in relation to increasing system load (e.g. Number of parallel users, number of transactions).
  - **Performance Testing**
    - Measurements of processing speed and response time for specific use cases, normally in relation to increasing load.

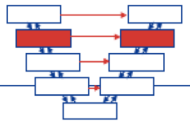
(continued on next page)

System Testing – Test Strategy
Non-functional Requirements (2)

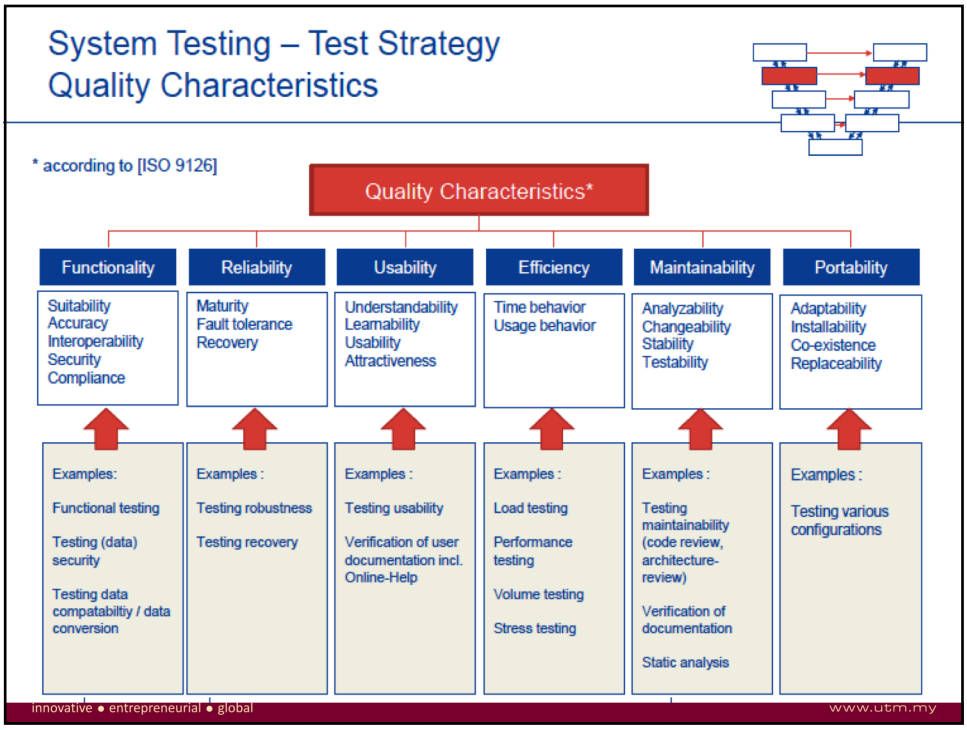


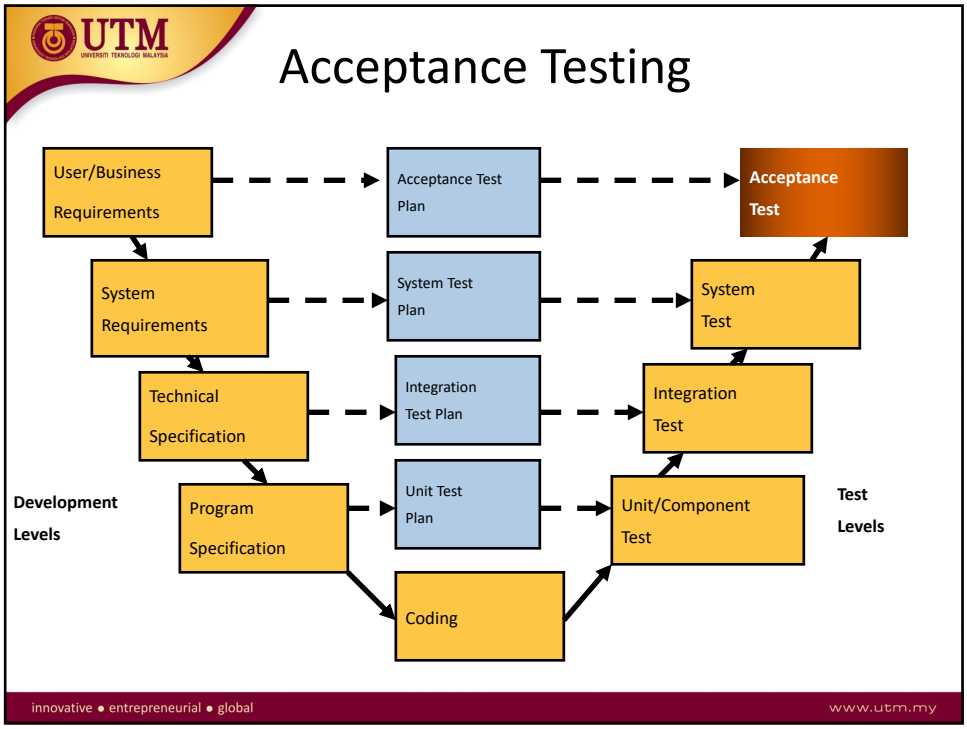
- **Volume testing**
  - Observation of system behavior in relation to the amount of data transmitted (e.g. processing large volumes of data).
- **Stress testing**
  - Observation of system behavior when overloaded.
- **Testing (data) security**
  - Test of access authorisations to the system or data.
- **Testing reliability**
  - During nonstop operation (e.g. amount of failures per hour with given user profile).
- **Testing robustness**
  - Against incorrect operation, programming errors, hardware failure etc. Tests also include system recovery handling.

System Testing – Test Strategy
Non-functional Requirements (3)



- **Testing compliance/conversion of data**
  - Verifying the compatibility of existing systems. Import/export of data base etc.
- **Testing different configurations**
  - of the system, e.g. different versions of operating systems, language, hardware platform etc.
- **Testing usability**
  - Verifying the suitability of operation, understandability, and of system output etc., in relation to the needs of the user.
- **Testing documentation**
  - Testing conformity with the system behavior (e.g. user manual)
- **Testing portability/maintainability**
  - Understandability and current content of development documents, modular system structure etc.


System Testing – Test Strategy
Quality Characteristics
* according to [ISO 9126]
Quality Characteristics*
Functionality
Reliability
Usability
Efficiency
Maintainability
Portability
Suitability
Accuracy
Interoperability
Security
Compliance
Maturity
Fault tolerance
Recovery
Understandability
Learnability
Usability
Attractiveness
Time behavior
Usage behavior
Analyzability
Changeability
Stability
Testability
Adaptability
Installability
Co-existence
Replaceability
Examples:
Functional testing
Testing (data) security
Testing data compatabiltiy / data conversion
Examples :
Testing robustness
Testing recovery
Examples :
Testing usability
Verification of user documentation incl. Online-Help
Examples :
Load testing
Performance testing
Volume testing
Stress testing
Examples :
Testing maintainability (code review, architecture-review)
Verification of documentation
Static analysis
Examples :
Testing various configurations
innovative • entrepreneurial • global
www.utm.my


UTM
Acceptance Testing
User/Business Requirements
Acceptance Test Plan
Acceptance Test
System Requirements
System Test Plan
System Test
Technical Specification
Integration Test Plan
Integration Test
Program Specification
Unit Test Plan
Unit/Component Test
Coding
Development Levels
Test Levels
innovative • entrepreneurial • global
www.utm.my

Acceptance Testing

Definition

- **Acceptance testing:** *Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.*

Acceptance Testing

Definition

- Usually the responsibility of the Customer/End user, though other stakeholders may be involved. Customer may sub-contract the Acceptance test to a third party
- Goal is to establish confidence in the system/part-system or specific non-functional characteristics (e.g. performance)
- Usually for ensuring the system is ready for deployment into production
- May also occur at other stages, e.g.
  - Acceptance testing of a COTS product before System Testing commences
  - Acceptance testing a component's usability during Component testing
  - Acceptance testing a new significant functional enhancement/middleware release prior to deployment into System Test environment.

User Acceptance Testing (UAT)

- Usually the final stage of validation
- conducted by or visible to the end user and customer
- testing is based on the defined user requirements
- Often uses the '**Thread Testing**' approach:
- *'A testing technique used to test the business functionality or business logic of the application in an end-to-end manner, in much the same way a User or an operator might interact with the system during its normal use.'*

  - Watkins 2001

  This approach is also often used for Functional Systems Test - The same Threads serve both test activities

User Acceptance Testing

- Often use a big bang approach
- black box testing techniques most commonly used
- Regression testing to ensure changes have not regressed other areas of the system

User Acceptance Testing

Testing Pearl of Wisdom

- ***"If love is like an extended software Q.A. suite, then true love is like a final Acceptance Test – one often has to be willing to endure compromise, bug fixes and work-arounds; otherwise, the software is never done "***

The Usenet Oracle

Operational Acceptance Testing (OAT)

- The Acceptance of the system by those who have to administer it.
- Features covered include:
  - testing of backup/restore
  - disaster recovery
  - user management
  - maintenance tasks
  - periodic checks of security vulnerabilities
- 'The objective of OAT is to confirm that the Application Under Test (AUT) meets its operational requirements, and to provide confidence that the system works correctly and is usable before it is formally "handed over" to the operation user. OAT is conducted by one or more Operations Representatives with the assistance of the Test Team' 1 –Watkins 2001

Operational Acceptance Testing (OAT)

- Employs a Black Box Approach for some activities
- Also employs a Thread Testing approach – Operations representatives performing typical tasks that they would perform during their normal usage of the system
- Also addresses testing of System Documentation, such as Operations manuals

Contract / Regulation Acceptance Testing

- Contract Acceptance Testing - testing against the acceptance criteria defined in the contract
  - final payment to the developer depends on contract acceptance testing being successfully completed
  - acceptance criteria defined at contract time are often imprecise, poorly defined, incomplete and out-of-step with subsequent changes to the application
- Regulation Acceptance testing is performed against any regulations which must be adhered to, such as governmental, legal or safety regulations

Alpha & Beta Testing

- early testing of stable product by customers/users
- feedback provided by alpha and beta testers
- alpha tests performed at developer's site by customer
- beta tests conducted at the customer site by end user/customer
- published reviews of beta release test results can make or break a product (e.g. PC games)

Other Acceptance Test Terms

- Factory Acceptance Testing (FAT)
- Site Acceptance Testing (SAT)
- Both address acceptance testing for systems that are tested before and after being moved to a customer's site