Chapter 2: Testing throughout the

Software Life Cycle

**2.1 Software Testing Terminology**

**Failure:** Failure means the inabilityof a system or component to perform a required function according to its specification.

**Fault/Bug/Defect:** is a condition that in actual causes a system to produce failure.

Bug

**Input Failure**

**Error:** Whenever a development team member makes a mistake in any phase of SDLC, errors are produced. It might be a typographical error, a misleading of a specification, a misunderstanding of what a subroutine does, and so on. An error causes a bug and the bug in turn causes failures.

Failures

Bug

Error

**Test Case:** A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application. A test case can have the following elements: Test Case ID, Test Data, Actual Result, Expected Result etc. **Testware:** The artifacts that are produced during the testing process which are required to plan, design, and execute tests is called Testware. Testware includes documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing. Generally, Testware is also called as Testing Tools.

**Incident:** When a failure occurs, it may or may not be readily apparent to the user. An incident is the symptom associated with the failure that alerts the user about the occurrence of a failure.

**Test Oracle:** An oracle is a mechanism for determining whether the program has passed or failed a test. The use of oracles involves comparing the output(s) of the system under test, for a given test-case input, to the output(s) that the oracle determines that product should have. The simple oracle is comparing actual result with expected results by hand. This can be very time- consuming, so automated oracles are sought. The oracle might be:

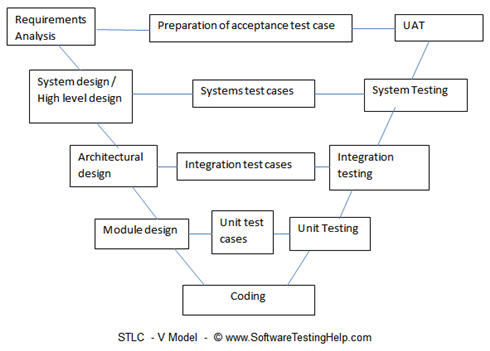
* a program (separate from the system under test) which takes the same input and produces the same output
* documentation that gives specific correct outputs for specific given inputs
* a documented algorithm that a human could use to calculate correct outputs for given inputs
* a human domain expert who can somehow look at the output and tell whether it is correct
* or any other way of telling that output is correct.

**2.2 Software Development Models**

**2.2.2 V-Model for Testing**

The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.



**Verification:**

Verification is done at the starting of the development process. It includes reviews and meetings, walkthroughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications. It makes sure that the product is designed to deliver all functionality to the customer. It answers the questions like: Am I building the product right? Am I accessing the data right (in the right place; in the right way).

**Verification phases:**

* **Requirements analysis:** In the requirements analysis phase, the first step in the verification process, the requirements of the system are collected by analyzing the needs of the user(s). Usually, the users are interviewed and a document called the user requirements document is generated. The user requirements document will typically describe the system's functional, interface, performance, data, security, etc. requirements as expected by the user.
* **System design:** Systems design is the phase where system engineers analyze and understand the business of the proposed system by studying the user requirements document. They figure out possibilities and techniques by which the user requirements can be implemented. The software specification document which serves as a blueprint for the development phase is generated. This document contains the general system organization, menu structures, data structures, entity diagrams, data dictionary etc. The documents for system testing are prepared.
* **Architecture design:** The phase of the design of computer architecture and software architecture can also be referred to as high-level design. The baseline in selecting the architecture is that it should realize all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology details etc. The integration testing design is carried out in the particular phase.[
* **Module design:** The module design phase can also be referred to as low-level design. The designed system is broken up into smaller units or modules and each of them is explained so that the programmer can start coding directly. The low level design document or program specifications will contain a detailed functional logic of the module, in pseudocode: database tables, with all elements, including their type and size, all interface details with complete API references, all dependency issues, complete input and outputs for a module. The unit test design is developed in this stage.

**Validation:**

Validation determines if the system complies with the requirements and performs functions for which it is intended and meets the organization’s goals and user needs. It is done at the end of the development process and takes place after verifications are completed. It answers the question like: Am I building the right product? Am I accessing the right data (in terms of the data required to satisfy the requirement). It is performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment. Validation determines the correctness of the final software product by a development project with respect to the user needs and requirements.

**Validation phases:**

* Unit testing
* Integration testing
* System testing
* User acceptance testing

**Advantages of V-model:**

* Simple and easy to use.
* Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
* Proactive defect tracking – that is defects are found at early stage.
* Avoids the downward flow of the defects.
* Works well for small projects where requirements are easily understood.

**Disadvantages of V-model:**

* Very rigid and least flexible.
* Software is developed during the implementation phase, so no early prototypes of the software are produced.
* If any changes happen in midway, then the test documents along with requirement documents has to be updated.

**When to use the V-model:**

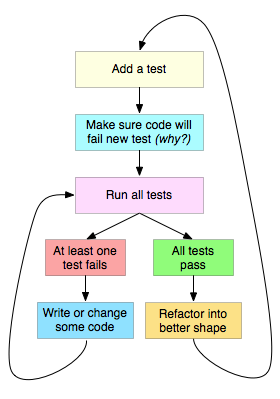
* The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
* The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

**2.2.3** **Iteration Model**

**Test Driven Development (TDD):** Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.

The following sequence of steps is generally followed:

* **Add a test:** In test-driven development, each new feature begins with writing a test. Write a test that defines a function or improvements of a function, which should be very succinct. To write a test, the developer must clearly understand the feature's specification and requirements. The developer can accomplish this through use cases and user stories to cover the requirements and exception conditions, and can write the test in whatever testing framework is appropriate to the software environment.
* **Run all tests and see if the new test fails:** This validates that the test case is working correctly, that the new test does not mistakenly pass without requiring any new code, and that the required feature does not already exist. This step also tests the test itself, in the negative: it rules out the possibility that the new test always passes, and therefore is worthless. The new test should also fail for the expected reason. This step increases the developer's confidence that the unit test is testing the correct constraint, and passes only in intended cases.
* **Write the code:** The next step is to write some code that causes the test to pass. The new code written at this stage is not perfect and may, for example, pass the test in an inelegant way. That is acceptable because it will be improved and honed in Step 5. At this point, the only purpose of the written code is to pass the test. The programmer must not write code that is beyond the functionality that the test checks.



* **Run tests:** If all test cases now pass, the programmer can be confident that the new code meets the test requirements, and does not break or degrade any existing features. If they do not, the new code must be adjusted until they do.
* **Refactor both new and old code to make it well structured:** The growing code base must be cleaned up regularly during test-driven development. New code can be moved from where it was convenient for passing a test to where it more logically belongs. Duplication must be removed. Object, class, module, variable and method names should clearly represent their current purpose and use, as extra functionality is added. As features are added, method bodies can get longer and other objects larger. They benefit from being split and their parts carefully named to improve readability and maintainability, which will be increasingly valuable later in the software lifecycle. By continually re-running the test cases throughout each refactoring phase, the developer can be confident that process is not altering any existing functionality.
* **Repeat:** Starting with another new test, the cycle is then repeated to push forward the functionality. The size of the steps should always be small, with as few as 1 to 10 edits between each test run. If new code does not rapidly satisfy a new test, or other tests fail unexpectedly, the programmer should undo or revert in preference to excessive debugging.

**\*\*\*NB. Got to following link for further example:** [1]

**2.3 Software Testing Methods, Level and Types**

**Testing Method:** There are several approaches / techniques of Software Testing [2]. These are:

* **Static Testing:** Static testing is the testing of the software work products manually, or with a set of tools, but they are not executed. It starts early in the Life cycle and so it is done during the verification process. It does not need computer as the testing of program is done without executing the program. For example: reviewing, walk through, inspection, etc.
* **Dynamic Testing:** Dynamic testing is the testing of the dynamic behavior of code. It involves working with the software, giving input values and checking if the output is as expected by executing specific test cases which can be done manually or with the use of an automated process. It is done during Validation process. The software is tested by executing it on computer. Ex: Unit testing, integration testing, system testing.
* **Black Box Testing:** Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it. It also known as Specification-based testing technique or input/output driven testing techniques because they view the software as a black-box with inputs and outputs.
* **White Box Testing:** White-box testing tests internal structures or workings of a program. It is also known as Structure-based or ‘glass-box’ testing technique because here the testers require knowledge of how the software is implemented, how it works. In white-box testing the tester is concentrating on how the software does it.

**Testing Level:** Each phase of SDLC goes through the testing. Hence there are various levels of testing.

* Unit Testing
* Component Testing
* Integration Testing
* System Testing
* Acceptance Testing
  + Alpha Testing
  + Beta Testing

**Testing Types:** A test type is focused on a particular test objective.

* Functional Testing
* Non-functional Testing
* Structural Testing
* Testing Related to Changes:
  + Re-Test
  + Regression Test

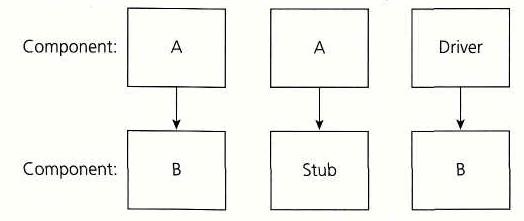
**2.4 Test Level:**

**2.4.1 Component/Unit Testing:**

**Stubs:** Stubs are dummy modules which are known as "called programs" which is used in integration testing (top down approach), used when sub programs are under construction.

**Drivers:** Drivers are also kind of dummy modules which are known as "calling programs", which is used in bottom up integration testing, used when main programs are under construction.

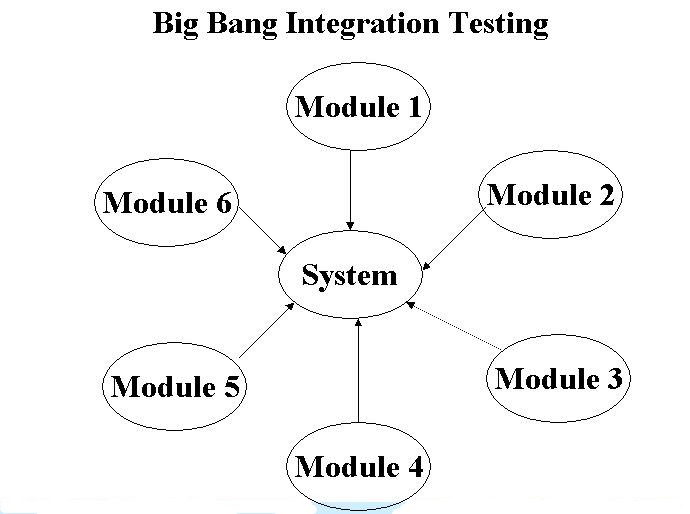
**Example:** Assume you have 2 modules, Module A and module B. Module A is ready and we need to test it, but module A calls functions from Module B which are not ready, so developer will write a dummy module which simulates B and returns values to module A. This dummy module code is known as stub.



Now suppose you have module B ready but module A which calls functions from module B is not ready so developer will write a dummy piece of code for module A which will return values to module B. This dummy piece of code is known as driver.

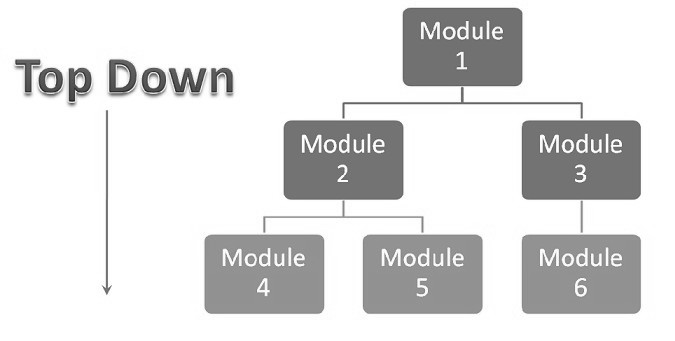
**2.4.2 Integration Testing:** Integration testing tests the interface between modules of the software application.  The different modules are first testing individually and then combined to make a system. Testing the interface between the small units or modules is integration testing. It is usually conducted by software integration tester and in continuation to the development.  There are different techniques available for integration testing:

1. **Big Bang Integration Testing:** In type of integration testing all the modules are combined first and then tested together. As per the below image all the modules from ‘Module 1′ to ‘Module 6′ are integrated simultaneously then the testing is carried out.

**Advantage:** Big Bang testing has the advantage that everything is finished before integration testing starts.

**Disadvantage:** The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.

1. **Top Bottom Integration Testing:** This type of testing takes place from top to bottom uses Stubs which are substitutes of components. The top module is tested first. Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu). Below is the diagram of ‘Top down Approach':

**Advantages of Top-Down approach: **

* The tested product is very consistent because the integration

testing is basically performed in an environment that almost similar to

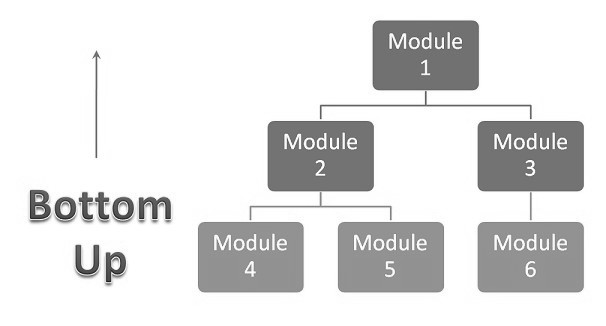
that of reality.

* Stubs can be written with lesser

time because when compared to the drivers then stubs are simpler to author.

**Disadvantages of Top-Down approach:** Basic functionality is tested at the end of cycle.

1. **Bottom to Top Integration Testing:** This type of testing take from bottom to top and uses Drivers which are substitutes of components. The bottom module is tested first. Testing takes place from the bottom of the control flow upwards. Below is the image of ‘Bottom up approach'.

**Advantages:** In this approach development and testing can be done together so that the product or application will be efficient and as per the customer specifications.

**Disadvantages**:

* We can catch the Key interface defects at the end of cycle.
* It is required to create the test drivers for modules at all levels except the top control

**Objective of Integration Testing:** Integration testing is performed to check the following points:

* To check whether the modules developed by individual developers when combined are according to standards and gives the expected results.
* When modules are combined, sometimes the data travelling between modules has many errors which may not give the expected results. So integration testing is performed to find the defects or bugs in all the interfaces.
* To check the integration between any third party tools used

**2.4.3 System Testing:**

System Testing is a type of black box testing technique thus the knowledge of internal code in not required. It is a high level testing always performed after integration testing. It considers the software application as a whole to check if the system is complaint with the user requirements. It would depend on the user or the organization to choose which type of system testing should be performed on the application.

**Objective of System Testing:** System testing is performed to check the following points:

* To check whether the software system is made according to the customer needs written in Software Requirements Specifications, it meets both functional and non-functional design requirements of the system.
* When all the modules are combined as a whole, many errors and facts may arise which may not give the expected results? So system testing is performed to find the defects or bugs in all the interfaces as well the whole system.
* To execute the real –life scenarios on the software. It is done on the staging server which is very much similar to the production server where the software would actually be deployed. The system test cases are made according to the end- to – end use perspective.

System testing can be broadly classified in two types:

1. **Functional Testing:** Functional testing is a quality assurance (QA) process and a type of black-box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing). Functional testing usually describes what the system does.

Functional testing typically involves six steps:

* The identification of functions that the software is expected to perform
* The creation of input data based on the function's specifications
* The determination of output based on the function's specifications
* The execution of the test case
* The comparison of actual and expected outputs
* To check whether the application works as per the customer need.

1. **Non-Functional Testing:** Non functional test is used to major **non-functional attributes of software systems.** Let’s take **non functional requirements** examples; in how much time does the software will take to complete a task? or how fast the response is. Here we have considered following testing as non-functional testing:

**#**[**Performance Testing**](http://www.softwaretestinghelp.com/introduction-to-performance-testing-loadrunner-training-tutorial-part-1/)**:**Evaluates the overall performance of the system. Key elements are as follows:

– Validate that the system meets the expected response time.  
– Evaluate that the significant elements of the application meets the desired response time.  
– It can also be conducted as a part of integration testing.

– It can also be conducted as a part of systems testing.

**#**[**Load Testing**](http://www.softwaretestinghelp.com/hp-loadrunner-load-testing-tool-training-tutorials/)**:** Evaluates whether the system’s performance is as expected under normal and expected conditions. Key points are

– Validate that the system performs as expected when concurrent users access the application and gets the expected response time.

– This test is repeated with multiple users to get the response time and throughput.  
– At the time of testing, the data base should be realistic.  
 – The test should be conducted on a dedicated server which stimulates the actual environment.

**#Stress Testing:** Evaluates whether the system’s performance is as expected when it is low on resources. Key points are:

– Test on low memory or low disc space on clients / servers that reveals the defects which cannot be found under normal conditions.

– Multiple user performance the same transactions on the same data.  
– Multiple clients connected to servers with different workloads.

**2.5 Maintenance Testing:**

Maintenance Testing is done on the already deployed software. The deployed software needs to be enhanced, changed or migrated to other hardware. The Testing done during this enhancement, change and migration cycle is known as maintenance testing.

Once the software is deployed in operational environment it needs some maintenance from time to time in order to avoid system breakdown, most of the banking software systems needs to be operational 24\*7\*365. So it is very necessary to do maintenance testing of software applications.

In maintenance testing, tester should consider 2 parts:

* Any changes made in software should be tested thoroughly.
* The changes made in software do not affect the existing functionality of the software, so regression testing is also done.

**Why is Maintenance Testing required**

User may need some more new features in the existing software which requires modifications to be done in the existing software and these modifications need to be tested.

End user might want to migrate the software to other latest hardware platform or change the environment like OS version, Database version etc. which requires testing the whole application on new platforms and environment.

\*\*\* For rest of the topics of **chapter 2**, please check the provided **ISTQB handout**.

**References:**

|  |  |
| --- | --- |
| [1] | "test driven development example," [Online]. Available: https://technologyconversations.com/2013/12/20/test-driven-development-tdd-example-walkthrough/. |
| [2] | "software testing methods levels and types," [Online]. Available: http://www.slideshare.net/confiz/software-testing-methods-levels-and-types . |