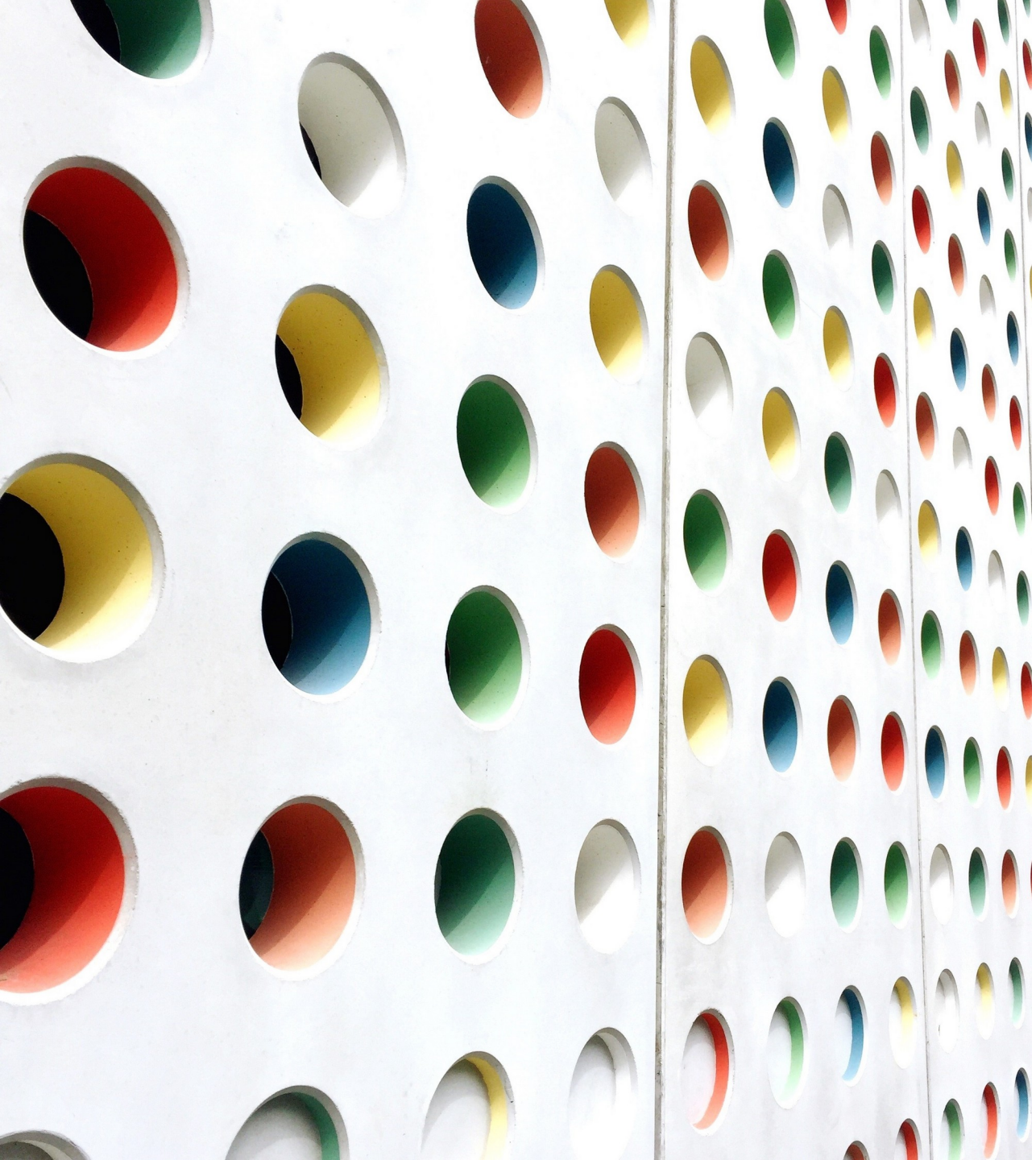


Software Modeling

5.1 translating requirement model into design model

We work to transform analysis **model into** four levels of **design** detail: the data structure, the system architecture, the interface representation, and the component level detail. During each **design** activity, we apply basic concepts and principles that lead to high quality of software.

Requirements modeling is an approach used in projects where the **requirements** keep on evolving throughout the project. ... This will help ensure that the project's exact **requirements** are documented clearly. By doing so, you can create a strong foundation for the project's overall needs and specifications

- The term **data model** can refer to two distinct but closely related concepts. Sometimes it refers to an abstract formalization of the objects and relationships found in a particular application domain:

- for example the customers, products, and orders found in a manufacturing organization. At other times it refers to the set of concepts used in defining such formalizations.
- The main aim of data models is to support the development of information systems by providing the definition and format of data.

Data modelling generally begins at a conceptual, abstractive stage where ideas are broken down into initial concepts. From there, the various broad components of the data system are created without any detail.

5.2 Analysis Modeling

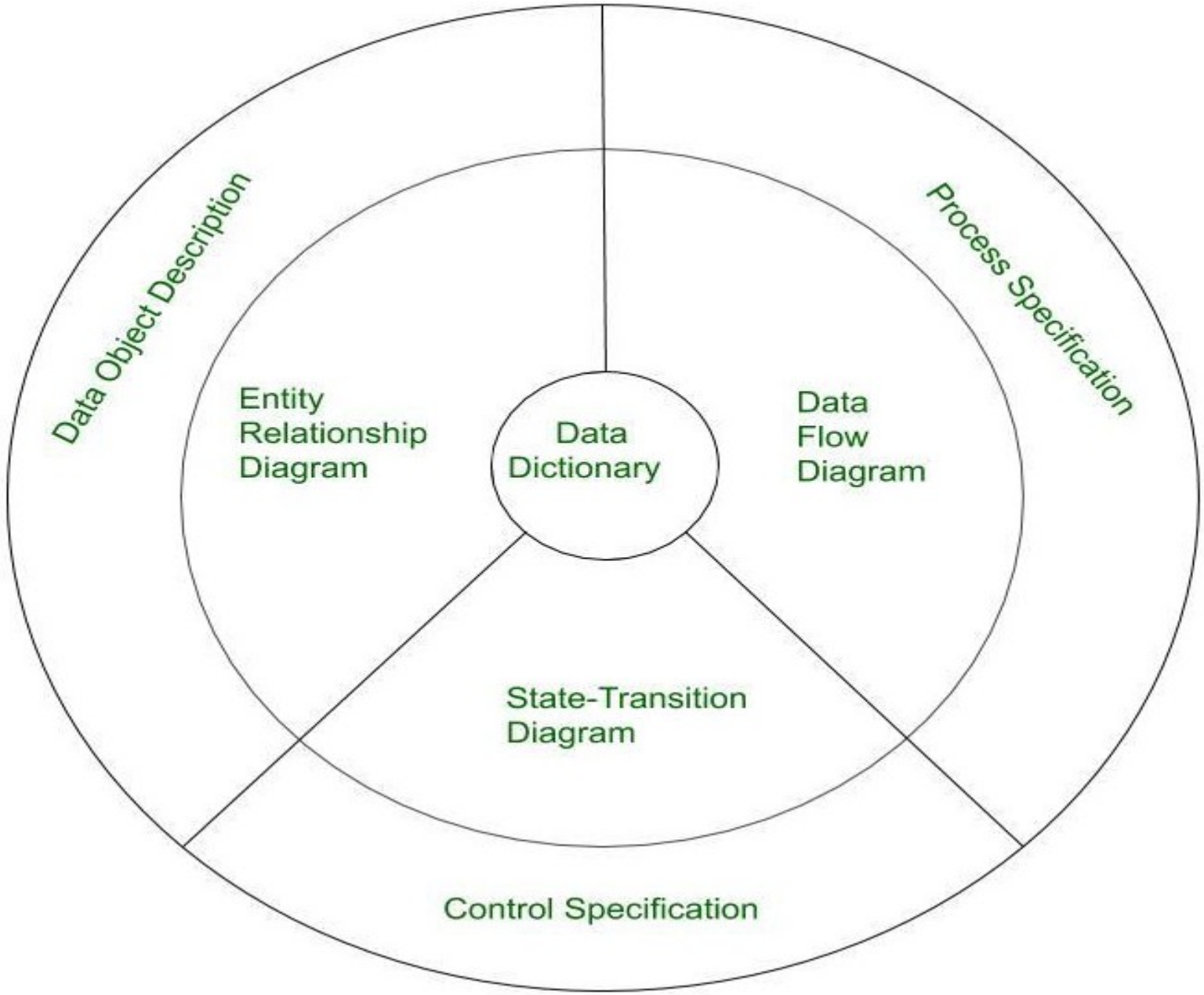
- **Analysis Model** is a technical representation of the system. It acts as a link between system description and design model.
- In Analysis Modelling, information, behavior and functions of the system is defined and translated into the architecture, component and interface level design in the design modeling.

Objectives of Analysis Modelling:

It must establish a way of creation of software design.

It must describe requirements of customer.

It must define set of requirements which can be validated, once the software is built.


Data Object Description
Process Specification
Entity
Relationship
Diagram
Data
Dictionary
Data
Flow
Diagram
State-Transition
Diagram
Control Specification

Elements of Analysis model

Data Dictionary:

It is a repository that consists of description of all data objects used or produced by software. It stores the collection of data present in the software. It is a very crucial element of the analysis model. It acts as a centralized repository and also helps in modelling of data objects defined during software requirements.

Entity Relationship Diagram (ERD):

It depicts relationship between data objects and used in conducting of data modelling activity. The attributes of each object in the Entity Relationship Diagram can be described using Data object description. It provides the basis for activity related to data design.

Data Flow Diagram (DFD):

It depicts the functions that transform data flow and it also shows how data is transformed when moving from input to output.

It provides the additional information which is used during the analysis of information domain and serves as a basis for the modeling of function.

It also enables the engineer to develop models of functional and information domain at the same time.

State Transition Diagram:
It shows various modes of behavior (states) of the system and also shows the transitions from one state to other state in the system. It also provides the details of how system behaves due to the consequences of external events. It represents the behavior of a system by presenting its states and the events that cause the system to change state. It also describes what actions are taken due to the occurrence of a particular event.

Process Specification:
It stores the description of each functions present in the data flow diagram. It describes the input to a function, the algorithm that is applied for transformation of input, and the output that is produced.

Control Specification:
It stores the additional information about the control aspects of the software. It is used to indicate how the software behaves when an event occurs and which processes are invoked due to the occurrence of the event. It also provides the details of the processes which are executed to manage events.

Data Object Description:
It stores and provides the complete knowledge about a data object present and used in the software. It also gives us the details of attributes of the data object present in Entity Relationship Diagram. Hence, it incorporates all the data objects and its attributes.

5.3 Design Modeling

- A **design model in software engineering** is an object-based picture or pictures that represent the use cases for a system.

- Or to put it another way, it's the means to describe a system's implementation and source code in a diagrammatic fashion. This type of representation has a couple of advantages.

Fundamental design concepts

Abstraction:-

Abstraction refers to a powerful design tool, which allows software designers to consider components at an abstract level.

Functional abstraction: This involves the use of parameterized subprograms. Functional abstraction can be generalized as collections of subprograms referred to as 'groups'.

Data abstraction: This involves specifying data that describes a data object. For example, the data object window encompasses a set of attributes (window type, window dimension) that describe the window object clearly.

Control abstraction: This states the desired effect, without stating the exact mechanism of control. For example, if and while statements in programming languages *(like C and C++)* are abstractions of machine code implementations, which involve conditional instructions.

INFORMATION HIDING:-

Modules should be specified and designed in such a way that the data structures and processing details of one module are not accessible to other modules. They pass only that much information to each other, which is required to accomplish the software functions.

Information hiding is of immense use when modifications are required during the testing and maintenance phase.

IEEE defines information hiding as 'the technique of encapsulating software design decisions in modules.

Structure:-

It refers to the structure of the system, which is composed of various components of a program/ system, the attributes (properties) of those components and the relationship amongst them. It enables the software engineers to analyze the software design efficiently.

Modularity:-

Modularity is achieved by dividing the software into uniquely named and addressable components, which are also known as modules. A complex system (large program) is partitioned into a set of discrete modules in such a way that each module can be developed independent of other modules

- **Concurrency:-**

- In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel.
- In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

- **verification** is the task of determining if the implementation of a **model** has been done correctly.
- Beyond program debugging, this means that **verification** data needs to be generated at various points in the **model** for comparison with expected values.

- **Aesthetics** :-

- is a core **design** principle that defines a **design's** pleasing qualities.

- In visual terms, **aesthetics** includes factors such as balance, color, movement, pattern, scale, shape and visual weight.

- Designers use **aesthetics** to complement their **designs'** usability, and so enhance functionality with attractive layouts.
- Aesthetics in software has the same role as in science in general - it is a powerful tool for motivation of the developer, a symptom of expert knowledge, a way to measure quality.

5.4 Design notations

- They are used when planning and should be able to communicate the purpose of a program without the need for formal code.

- Commonly used **design notations** are: Pseudocode. Flow charts.

- A **Data Flow Diagram** (**DFD**) is a traditional visual representation of the information flows within a system.

- A neat and clear **DFD** can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both


CUSTOMER
REGISTER
ORDER/
UPDATE
ORDER
ACK.
BILL
RECEIPT
OFFERS
SUPPLY
PRODUCT
ONLINE SHOPPING
SYSTEM
NEW
ORDER
ORDER
ACK.
REPORTS
NEW OFFERS/
DISCOUNT
FEEDBACK
PAYMENT
SUPPLY
MANAGEMENT
ACCOUNTS
SECTION

structured flowchart :-

essentially helps the mission to create new algorithms by encapsulating a range of data points inside an inter-linked illustration.

However, the flawless creation of such a diagram demands that software designers create a top-level flowchart that clearly reflects the problem.

A **flowchart** is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

➢ Decision tables :

- **Decision table** is a brief visual representation for specifying which actions to perform depending on given conditions. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.

- A decision table is a good way to settle with different combination inputs with their corresponding outputs and also called cause-effect table. Reason to call cause-effect table is a related logical diagramming technique called cause-effect graphing that is basically used to obtain the decision table.

5.5 Testing

Software Testing Methodology is defined as strategies and testing types used to certify that the Application Under Test meets client expectations.

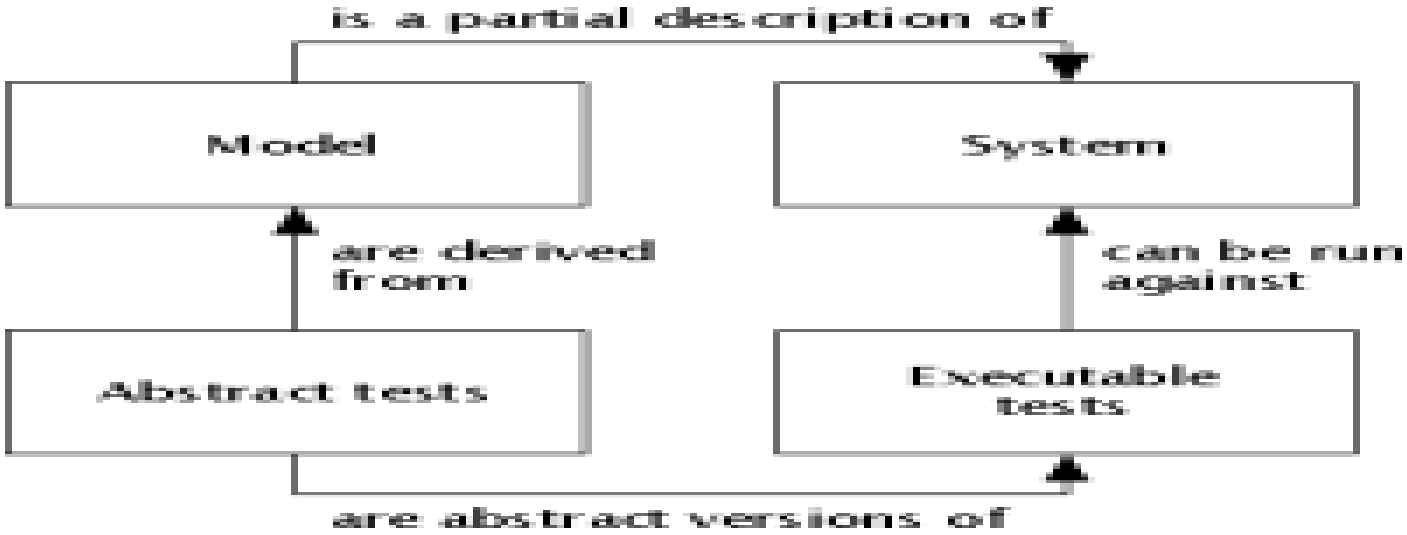
Hence Testing Methodologies could also refer to Waterfall, Agile and other QA models as against the above definition of Testing Methodologies.

Purpose

Description: Software testing is the process of verifying a system with the purpose of identifying any errors, gaps or missing requirement versus the actual requirement. Software testing is broadly categorized into two types - functional testing and non-functional testing.

Model-based testing is an application of model-based design for designing and optionally also executing artifacts to perform software testing or system testing.

Models can be used to represent the desired behavior of a system under test (SUT), or to represent testing strategies and a test environment.



Testing Methods:

Black-Box:

Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input, and observes the output generated by the system under test.

Black box testing is a powerful testing technique because it exercises a system end-to-end.

Black-box testing is a method of software testing that examines the functionality of an application without peering into its internal structures or workings. It is sometimes referred to as specification-based testing.

- Black-box testing, also called behavioral testing, focuses on the functional requirements of the software. That is, black-box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
- Black-box testing is not an alternative to white-box techniques. Rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.
- Black box testing is applied in final stages of testing.

- Black-box testing attempts to find errors in the following categories:

1. incorrect or missing functions
2. interface errors
3. errors in data structures or external data base access
4. behavior or performance errors, and
5. initialization and termination errors

White-Box:

- White-box testing, sometimes called glass-box testing
- It is a test case design method that uses the control structure of the procedural design to derive test cases.
- These control structures are described as component level design to derive test cases .
- Characteristics :

1) Guarantee that all independent paths within a module have been exercised at least once
2) Exercise all logical decisions on their true and false sides
3) Execute all loops at their boundaries and within their operational bounds, and
4) Exercise internal data structures to ensure their validity.

Level of Testing

Unit Test:

In computer programming, unit testing is a software testing method by which individual units of source code—sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures—are tested to determine whether they are fit for use.

Writing and maintaining unit tests can be made faster by using parameterized tests. These allow the execution of one test multiple times with different input sets, thus reducing test code duplication

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct

Integration Testing

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group.

Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.

is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface.

The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Big-Bang Integration Testing –
It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing.

In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems.

If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated.

Bottom-Up Integration Testing –
In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested.

The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem.

This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

Top-Down Integration Testing –
Top-down integration testing technique used in order to simulate the behavior of the lower-level modules that are not yet integrated.

In this integration testing, testing takes place from top to bottom.

First high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Mixed Integration Testing –
A mixed integration testing is also called sandwiched integration testing.

A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested.

In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches.

A mixed integration testing is also called sandwiched integration testing.

User Acceptance

User Acceptance Testing (UAT) is the final stage of any software development life cycle. This is when actual users test the software to see if it is able to carry out the required tasks it was designed to address in real-world situations.

UAT testers aim to validate changes that were made against original requirements.

also known as beta or end-user testing, is defined as testing the software by the user or client to determine whether it can be accepted or not.

This is the final testing performed once the functional, system and regression testing are completed.

This is typically the last step before the product goes live or before the delivery of the product is accepted.

5.6 Test Documentation

Test case Template:

A test case template is a document containing an organized list of test cases for different test scenarios that check whether or not the software has the intended functionality.

Other than the executable steps, a test case also contains preconditions for testing, test data provided, expected outcome, and actual result.

Software test case templates are blank documents that describe inputs, actions, or events, and their expected results, in order to determine if a feature of an application is working correctly.

Test case templates contain all particulars of test cases.

Test plan

A test plan is a document detailing the objectives, resources, and processes for a specific test for a software or hardware product.

The plan typically contains a detailed understanding of the eventual workflow.

A test plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements.

A test plan is usually prepared by or with significant input from test engineers.

Test plan document formats can be as varied as the products and organizations to which they apply.

Introduction to defect report

DEFECT REPORT, also known as Bug Report, is a document that identifies and describes a defect detected by a tester.

The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.

A defect report documents an anomaly discovered during testing. It includes all the information needed to reproduce the problem, including the author, release/build number, open/close dates, problem area, problem description, test environment, defect type, how it was detected, who detected it, priority, severity, status, etc.

The first step is to define the defect by writing a summary in the defect title and providing a general description of the problem.

Test Summary Report

The test summary report outlines the summation of software testing activities and final testing results.

Software testers are required to communicate testing results and findings to project stakeholders on the completion of a testing cycle.

The definition of a Test Summary is as simple as the name suggests.

Also known as a Test Closure Report, it provides stakeholders with a condensed account of the overall test results, defects and connected data following a test project.

Test report is an assessment of how well the Testing is performed.

Test Report is a document which contains a summary of all test activities and final test results of a testing project.