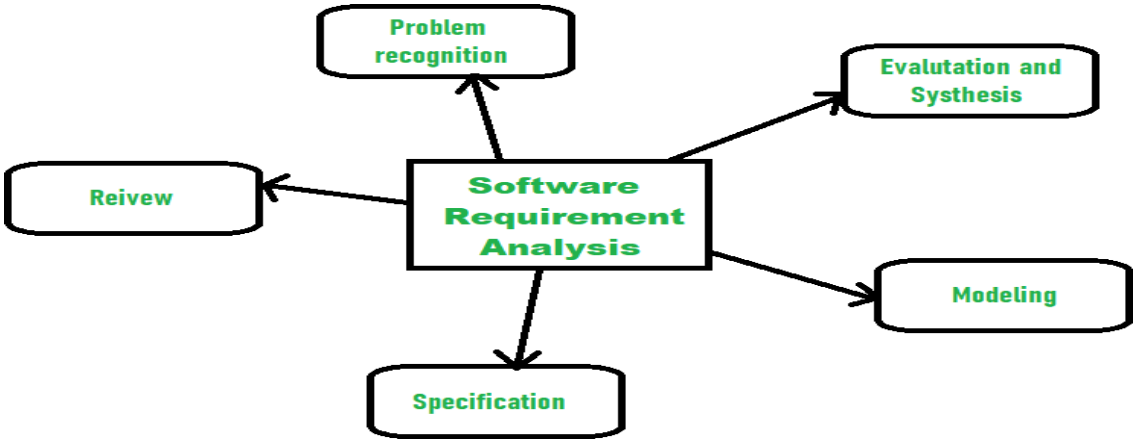
Requirement Identification

Software requirement means requirement that is needed by software to increase quality of software product. These requirements are generally a type of expectation of user from software product that is important and need to be fulfilled by software. Analysis means to examine something in an organized and specific manner to know complete details about it.

Therefore, Software requirement analysis simply means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem. There are several activities involved in analyzing Software requirements. Some of them are given below :



Problem Recognition :

The main aim of requirement analysis is to fully understand main objective of requirement that includes why it is needed, does it add value to product, will it be beneficial, does it increase quality of the project, does it will have any other effect. All these points are fully recognized in problem recognition so that requirements that are essential can be fulfilled to solve business problems.

Evaluation and Synthesis :

Evaluation means judgement about something whether it is worth or not and synthesis means to create or form something. Here are some tasks are given that is important in the evaluation and synthesis of software requirement :

- To define all functions of software that necessary.
- To define all data objects that are present externally and are easily observable.
- To evaluate that flow of data is worth or not.
- To fully understand overall behavior of system that means overall working of system.
- To identify and discover constraints that are designed.
- To define and establish character of system interface to fully understand how system interacts with two or more components or with one another.

Modeling :

After complete gathering of information from above tasks, functional and behavioral models are established after checking function and behavior of system using a domain model that also known as the conceptual model.

Specification :

The software requirement specification (SRS) which means to specify the requirement whether it is functional or non-functional should be developed.

Review :

After developing the SRS, it must be reviewed to check whether it can be improved or not and must be refined to make it better and increase the quality.

Test Table Requirements

A Decision Table is a table that shows the relationship between inputs and rules, cases, and test conditions. It's a very useful tool for both complicated software testing and requirements management. The decision table allows testers to examine all conceivable combinations of requirements for testing and to immediately discover any circumstances that were overlooked. True(T) and False(F) values are used to signify the criteria.

What is Decision Table Testing?

Decision table testing is a type of software testing that examines how a system responds to various input combinations. This is a methodical methodology in which the various input combinations and the accompanying system behavior (Output) are tabulated. That's why it's also known as a Cause-Effect table because it captures both causes and effects for improved test

Example 1 − How to Create a Login Screen Decision Base Table

Let's make a login screen with a decision table. A login screen with E-mail and Password Input boxes.

The condition is simple − The user will be routed to the homepage if they give the right username and password. An error warning will appear if any of the inputs are incorrect.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Legend

T - Make sure your login and password are correct.

F - Incorrect login or password

E - An error message appears.

H - The home screen appears.

Interpretation

Case 1 − Both the username and password were incorrect. An error message is displayed to the user.

Case 2 − The username and password were both right, however, the password was incorrect. An error message is displayed to the user.

Case 3 − Although the username was incorrect, the password was accurate. An error message is displayed to the user.

Case 4 − The user's username and password were both accurate, and the user went to the homepage.

We may generate two situations when converting this to a test case.

Enter the right username and password and click Login; the intended consequence is that the user will be sent to the homepage.

And one from the situation below.

- If the user types in the erroneous username and password and then clicks Login, the user should receive an error message.
- When you provide the proper username and incorrect password and click Login, the user should see an error message.
- If the user types the erroneous username and password and then clicks Login, the user should receive an error message.

Modeling a test design process

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels of phases of design:

- Interface Design
- Architectural Design
- Detailed Design

Elements of a System:

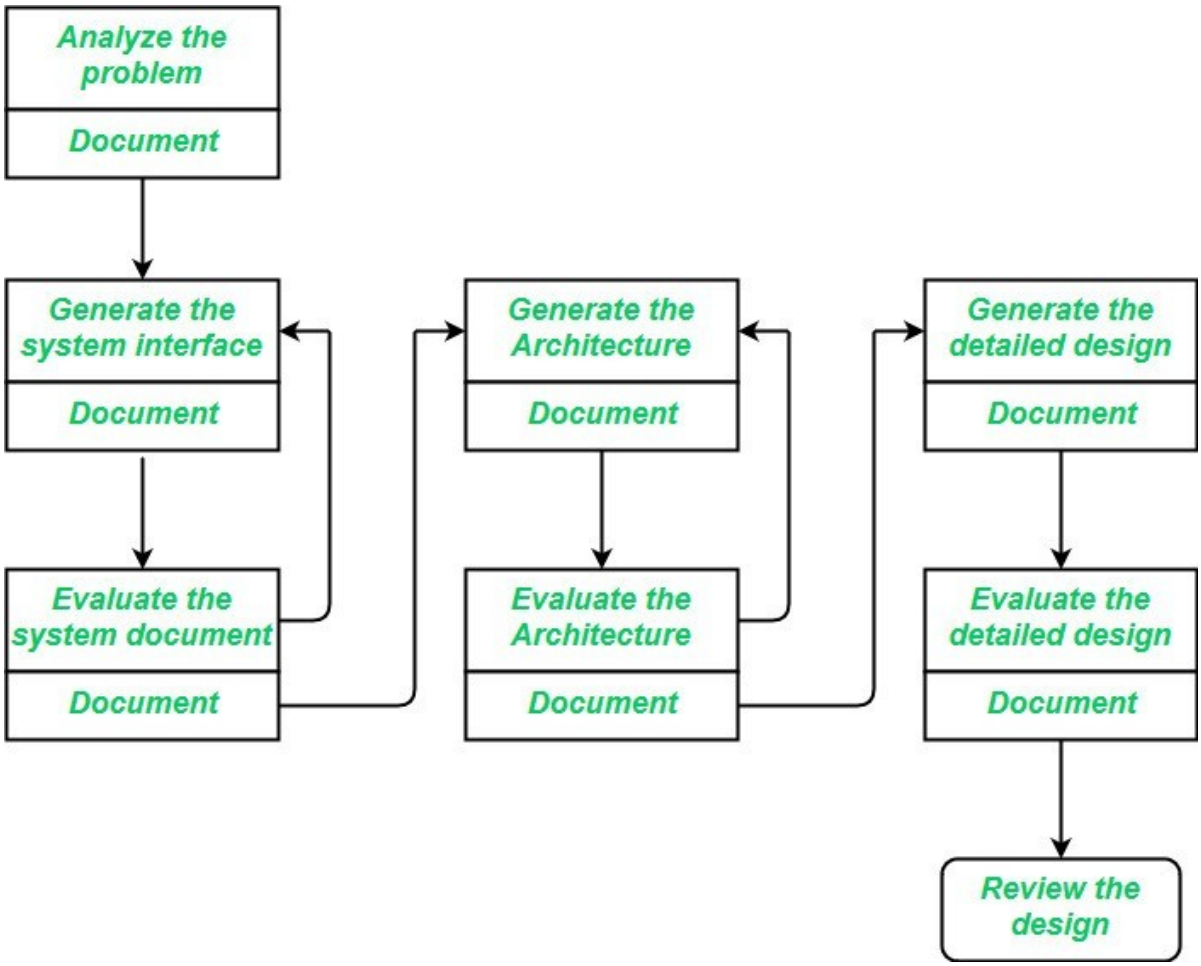
Architecture – This is the conceptual model that defines the structure, behavior, and views of a system. We can use flowcharts to represent and illustrate the architecture.

Modules – These are components that handle one specific task in a system. A combination of the modules makes up the system.

Components – This provides a particular function or group of related functions. They are made up of modules.

Interfaces – This is the shared boundary across which the components of a system exchange information and relate.

Data – This is the management of the information and data flow.



Interface Design: Interface design is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called agents. Interface design should include the following details:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification of the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

Architectural Design: Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored. Issues in architectural design includes:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

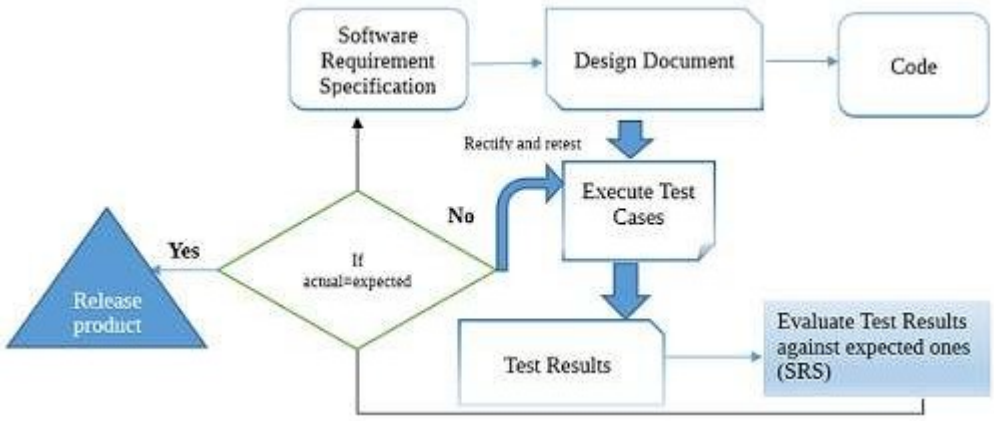
Detailed Design: Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures. The detailed design may include:

- ➢ Decomposition of major system components into program units.
- ➢ Allocation of functional responsibilities to units.
- ➢ User interfaces
- ➢ Unit states and state changes

- ➢ Data and control interaction between units
- ➢ Data packaging and implementation, including issues of scope and visibility of program elements
- ➢ Algorithms and data structures

Modeling Test Results

Test results are the outcome of the whole process of software testing life cycle. The results thus produced, offer an insight into the deliverables of a software project, significant in representing the status of the project to the stakeholders.



Few concepts:

Testing: It is a process of identifying whether a bug/error hides within a project and also assess the impact of the observation. It forms an important activity of Quality Assurance.

Debugging: This method involves identification and then correction of bugs/errors. When developers come across an error in the code, they resort to debugging. Debugging is thus a part of unit testing.

Test Case:Test Case is a document that consists of test data, preconditions, postconditions and expected results that are developed for a specific kind of test scenario, intended for serving a particular purpose.

Test Suite:It is a collection of test cases, that are aimed at testing a software application to detect that the application adheres to the requirement specifications. It basically consists of a detailed set of instructions to attain a common goal.

There are majorly two broad categories/types of testing, which helps to obtain the test results in the most appropriate way. They are as follows :

Manual Testing- It is the process of testing which comprises of a group of testers who examines the code for the presence of a bug. The testing is performed without the use of any tool. The tester tests the application just as an end-user would do, in order to find out defects, if any.

Automation Testing- Automating the test process is done with the help of a script or tool. A piece of code is used to detect a bug/error.

Considering the aforementioned categories of testing, how should one come to a conclusion as to which testing one must adopt to attain correct results. The following key points must be considered while deciding upon a specific type of testing :

- ➢ Automation Testing is a life saver when the project under consideration is quite large and complex in nature.
- ➢ When we need to repetitively test some part of the code, very often.
- ➢ When the requirements are quite stable, that is, requirements are not prone to change.
- ➢ When the application need to go through load and performance tests with many virtual users involved in it.

Boundary Value Testing

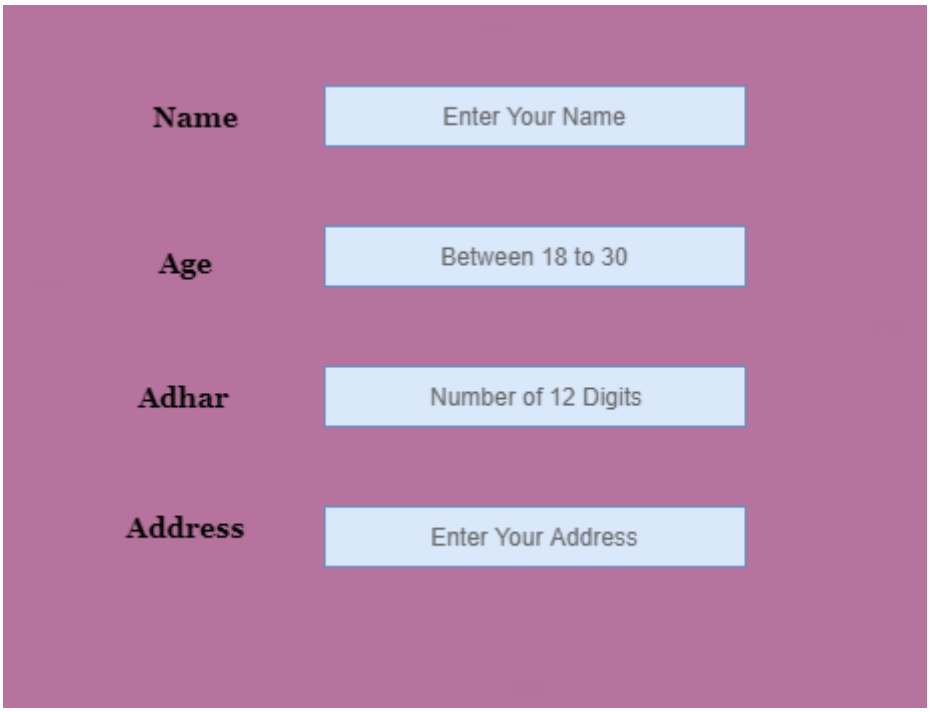
Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable. Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.



Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

Let's understand via practical:

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

Advantages of Boundary Value Analysis

1. Using BVA reduces the number of test cases required to cover the input domain by only testing a few values at each boundary instead of all possible ones. This saves time and resources and makes test cases more manageable and maintainable.

2. BVA can also help you find deviation errors error, overflow errors, or boundary condition errors that you would otherwise miss. This overall improves software quality.

3. BVA can find bugs that can affect application functionality, performance, or security, thus improving the quality and reliability of your software.

4. BVA ensures that the software can process the input values correctly.

5. Errors can be detected early in the software development cycle. This reduces the cost of fixing errors later.

Disadvantages of Boundary Value Analysis

1. The success of testing using this technique depends on the equivalence classes identified. This also depends on the tester's experience and application knowledge. Thus, incorrect identification of equivalence classes leads to incorrect limit testing. Equivalence classes refer to a software testing technique where input data is divided into groups or sets that are expected to exhibit similar behavior in the software system being tested. Each group is known as an equivalence class

2. Applications with open or missing one-dimensional boundaries are unsuitable for this technique. Other black-box techniques, such as "domain analysis", are used in such cases.

3. BVA may fail to identify defects that occur within the boundaries themselves. This can lead to false negatives and prevent developers from detecting important bugs early in development.

4. BVA may be effective at identifying potential edge cases, testing all possible boundary values in a given system may not be practical or feasible in every case.

Equivalence Class Testing

Equivalence class testing is better known as Equivalence Class Partitioning and Equivalence Partitioning. This is a renowned testing approach among all other software testing techniques in the market that allows the testing team to develop and partition the input data for analyzing and testing

and based on that the software products are partitioned and divided into number of equivalence classes for testing.

The equivalence classes that are divided perform the same operation and produce same characteristics or behavior of the inputs provided.

The test cases are created on the basis on the different attributes of the classes and each input from the each class is used for execution of test cases, validating the software functions and moreover validating the working principles of the software products for the inputs that are given for the respective classes.

It is also referred as the logical step in functional testing model approach that enhances the quality of test classes and by removing any redundancy or faults that can exist in the testing approach.

Types of Equivalence Class Testing

Weak Normal Testing Class: This type of testing uses only a single variable from each equivalence class during test cases. The word weak signifies single fault and in the testing scenario, there is only one element. The tester identifies the values in a systematic way.

Strong Normal Testing Class: This type of testing is associated with multiple fault assumptions and test cases are required for each element from equivalence class and the testing team covers the whole equivalence class by using every possible inputs.

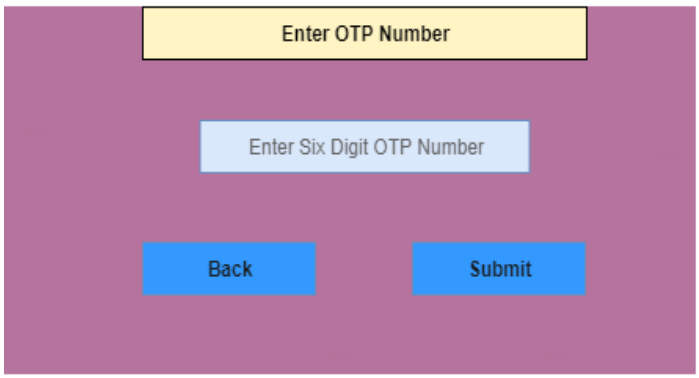
Weak Robust Testing Class: This type of testing results into single fault as it is weak and the expected output from invalid test cases cannot be defined. The testers usually spend a huge amount of time for defining the expected output from the test cases. The testing team mainly focus on testing the test cases for invalid values.

Strong Robust Testing Class: This form of class testing is redundant. So multiple fault assumptions are present and the equivalence classes are measured in the terms of valid and invalid inputs from test cases. However, for the testing team it is not feasible for reducing the redundancy.

Examples of Equivalence Partitioning technique

Assume that there is a function of a software application that accepts a particular number of digits, not greater and less than that particular number. For example, an OTP number which contains only six digits, less or more than six digits will not be accepted, and the application will redirect the user to the error page.

1. OTP Number = 6 digits

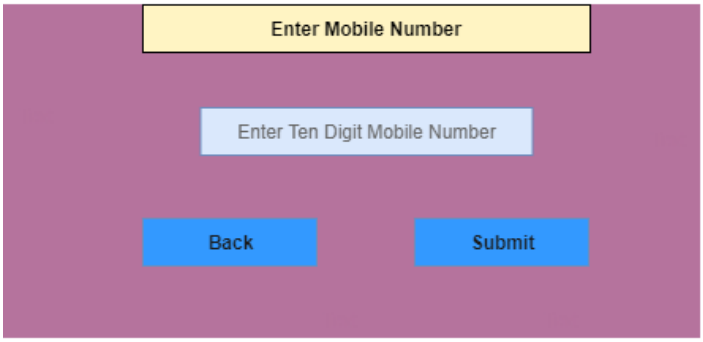


INVALID 1 Test case	INVALID 2 Test case	VALID 3 Test case	VALID
DIGITS >=7	DIGITS<=5	DIGITS = 6	DIGITS = 6
93847262	9845	456234	451483

Let's see one more example.

A function of the software application accepts a 10 digit mobile number.

2. Mobile number = 10 digits



INVALID 1 Test case	INVALID 2 Test case	VALID 3 Test case	VALID
DIGITS >=11	DIGITS<=9	DIGITS = 10	DIGITS =10
93847262219	984543985	9991456234	9893451483

In both examples, we can see that there is a partition of two equally valid and invalid partitions, on applying valid value such as OTP of six digits in the first example and mobile number of 10 digits in the second example, both valid partitions behave same, i.e. redirected to the next page.

Another two partitions contain invalid values such as 5 or less than 5 and 7 or more than 7 digits in the first example and 9 or less than 9 and 11 or more than 11 digits in the second example, and on applying these invalid values, both invalid partitions behave same, i.e. redirected to the error page.

We can see in the example, there are only three test cases for each example and that is also the principal of equivalence partitioning which states that this method intended to reduce the number of test cases.

Advantages:

- Equivalence class testing helps reduce the number of test cases, without compromising the test coverage.
- Reduces the overall test execution time as it minimizes the set of test data.
- It can be applied to all levels of testing, such as unit testing, integration testing, system testing, etc.
- Enables the testers to focus on smaller data sets, which increases the probability to uncovering more defects in the software product.
- It is used in cases where performing exhaustive testing is difficult but at the same time maintaining good coverage is required.

Disadvantages:

- It does not consider the conditions for boundary value.
- The identification of equivalence classes relies heavily on the expertise of testers.
- Testers might assume that the output for all input data set are correct, which can become a great hurdle in testing.