Contents

1. Requirements

A requirement represents a statement provided by a stakeholder about what they believe they need in order to solve a particular business problem or respond to a specific business need.

1.1 Requirement types

Requirement types are logical groupings of requirements by common functions, features and attributes. There are four requirement types within three distinct requirement levels:

1.1.1 Business Requirements Level

Business Requirement Type.

The business requirement is written from the Sponsor's point-of-view. It defines the objective of the project (goal) and the measurable business benefits for doing the project. The following sentence format is used to represent the business requirement and helps to increase consistency across project definitions: "The purpose of the [project name] is to [project goal -- that is, what is the team expected to implement or deliver] so that [measurable business benefit(s) -- the sponsor's goal]."

Business Requirement Example: “Build a family home to replace the home that was burnt down including the addition of a garage. “

1.1.2 User Requirements Level

User Requirement Type.

The user requirements are written from the user's point-of-view. User requirements define the information or material that is input into the business process, and the expected information or material as outcome from interacting with the business process (system), specific to accomplish the user's business goal. The following sentence format is used to represent the user requirement: "The [user role] shall [describe the interaction (inputs and outputs of information or materials) with the system to satisfy the user's business goal.]" or "The [user role] shall provide (input)/ receive (output.)"

User Requirement Example: “We need a family house with four bedrooms so that each child has their own bedroom”

1.1.3 Solution Requirements Level

Solution Requirement Example 1: “I want my bedroom to be painted pink so that everyone will know it is my room” – Stakeholder who raised this requirement is the little girl.

Solution Requirement Example 2: “I want my bedroom floor space to be at least 30 square meters so that I can practice my skateboard tricks in the bedroom” – Stakeholder who raised this requirement is the teenage boy in the family.

Solution Requirement Example 3: "Every bedroom must have an air-conditioning unit implemented so that the family can stay cool during the summer" – Stakeholder who raised this requirement is the father in consultation with the architect.

Solution Requirement Example 4: "The house must have fire resistant insulation in all the walls of the house to prevent significant fire damage." – Stakeholder who raised this requirement is the builder who is adhering to a regulatory requirement.

The solution requirements describe how the stakeholder wants to implement their stakeholder requirements. In this example, the stakeholder requirement relating to having four bedrooms has been expanded upon with more specific solution requirements describing how that stakeholder requirement must be implemented.

There are two sub-types of Solution Requirements:

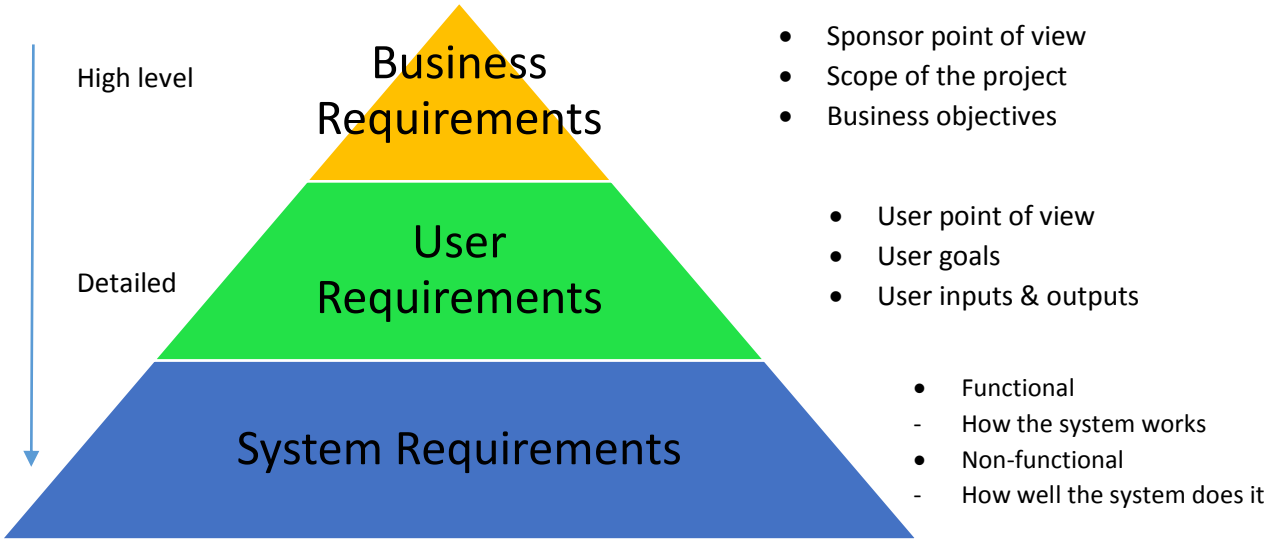
Functional Requirement Type.

This type of solution requirement describes how the solution must behave. In the example of the house, it describes how the house must look (colors, size of bedroom) and perform (have an air-conditioning unit in each bedroom). In a system related scenario example, the different functions that you want a system to perform is typically described as functional requirements (in an Agile Project context, it is referred to as a 'user story'). An easy way to remember this type of solution requirement is to think about what you want to the system to be able to do. Another example in the context of the house would be that a functional requirement exists to have internal doors, which can be opened and closed but not locked. This is something that you want the house to be able to do, a function you want the house to be able to perform.

Nonfunctional Requirement Type.

The non-functional requirement type of solution requirement describes the characteristics that you want the system to have. In the context of the house example, solution requirement 4 is describing a characteristic that is required of the house walls. It is not a function of the house but rather a characteristic of the walls. In the scenario of a system an example that compares to this house analogy would be a non-functional requirement describing the need to have a backup-system installed to be used in the event of a disaster to prevent unnecessary data loss. The non-functional type of solution requirement therefore describes the attributes a system or process should possess and not a function that the system must perform.

Source: http://business-analysis-excellence.com/types-of-requirements/



1.2 Requirement analysis

Requirements analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users.

Requirements analysis is critical to the success of a development project. Requirements must be actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design. Requirements can be functional and non-functional.

Conceptually, requirements analysis includes three types of activity:

- Eliciting requirements: the task of communicating with customers and users to determine what their requirements are. This is sometimes also called requirements gathering.
- Analyzing requirements: determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory, and then resolving these issues.
- Recording requirements: Requirements might be documented in various forms, such as natural-language documents, use cases, user stories, or process specifications.

1.2.1 Know Stakeholders' Requirements

You should compile an exhaustive list of the requirements of each stakeholder and end-user; you should compile all their requirements to get an overall picture.

- Give an exact picture of the limits and extent of the project to keep the requirements within the range and pertaining to the project alone.
- You can hold individual interviews as well as group discussions (requirements workshops) to discuss the requirements

- There are other techniques for eliciting the requirements like use cases, prototyping, data flow diagrams, and competitor analysis. It is essential that the exact requirements of the stakeholders are established.
- Build a prototype of the project to give an exact idea of the final results of the product or project to stakeholders.

1.2.2 Classify the Requirements

With so many requirements on the agenda, it will make better sense to group the requirements under various categories. There can be 3-4 types, such as:

- What requirements identify with functions and components the end-users are expecting?
- What requirements identify with the operational activities that need to be done?
- What requirements identify with the technical details needed for smooth functioning?
- What may be needed for the successful completion of the project?

1.2.3 Analyze the Requirements

Now it is necessary to go in depth about the nature of the requirements. You should determine whether the compiled list of requirements are clear in their purpose and are pertaining to the project or the process. Is there any ambiguity inherent? Are there any contradictory interests to other issues? Is implementation of each requirement feasible?

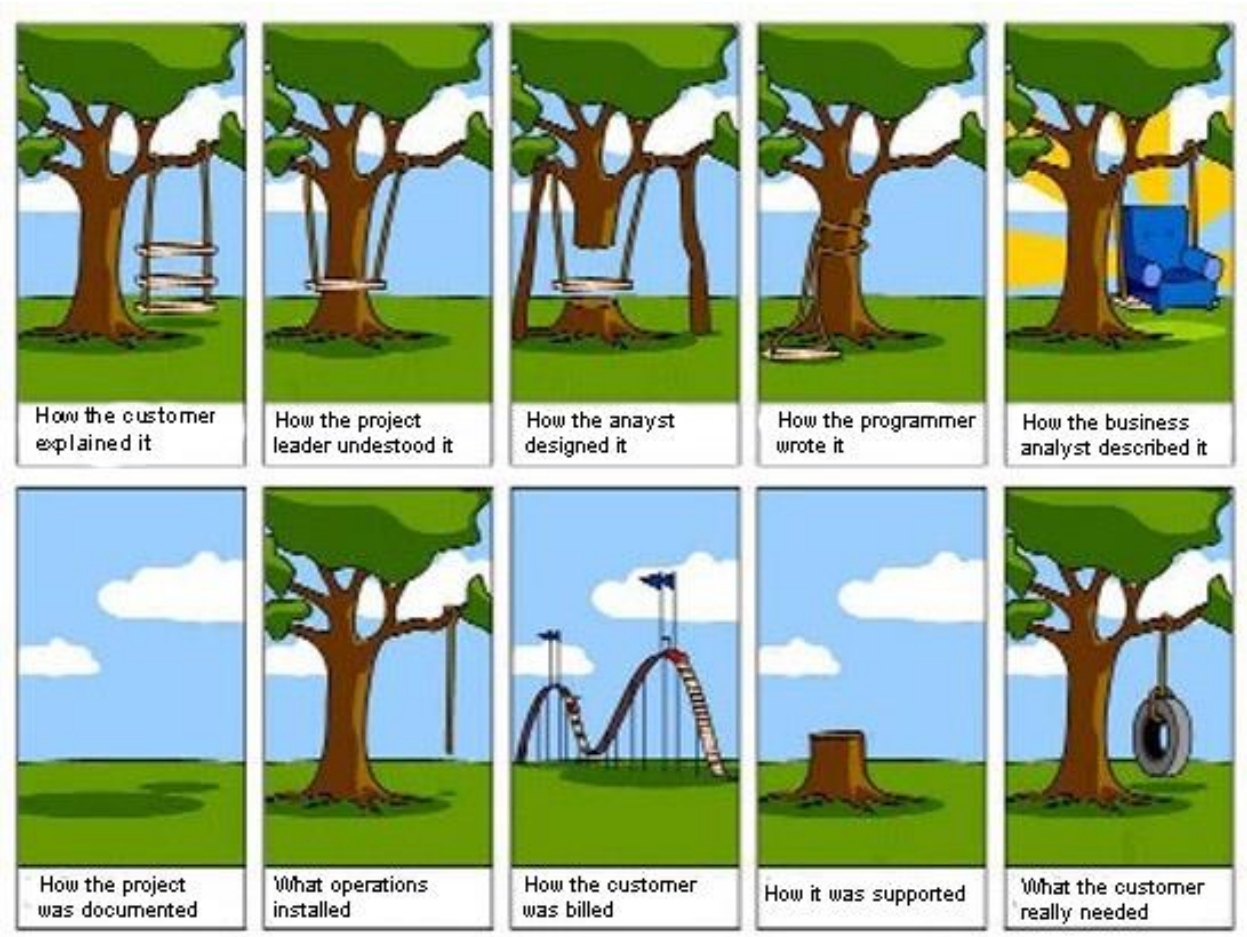
- List all the requirements with regard to priority and relevancy to the project.
- Also try to predict the impact of any changes proposed.
- Solve the ambiguous and conflicting details that have come up.
- The final list of requirements must be clear, unambiguous, concise, feasible, and relevant to the project.

1.2.4 Document the Requirements

Once the requirements are completely known and the stakeholders/end-users are clear about what they want from the project, what they are going to achieve, and they have seen the prototype and are satisfied, it is time to create a document that will combine all the details and get it signed by all stakeholders/end-users and the project manager. This will be the rule book for the project. All stakeholders, end-users, project personnel, and developers should be given a copy to apprise them of the project goals.

An efficiently done business requirements analysis will enable you to pinpoint exactly what is wanted from the project and how you can achieve it. Once this is done, there will be no ambiguity about the diverse requirements/specifications connected with the project and there will be a focused and well planned execution of the project with no chance for a scope or function creep.

Requirements are proven complete to the end user by creating test cases that cover them. Test cases are described in the next chapter.



2. Test cases

Overview

A test case is used to examine whether the system displays the desired behavior under specific circumstances. It must therefore contain all of the ingredients to cause that system's behavior and determine whether or not it is correct. A well-known way to describe system behavior is 'Input → Processing → Output'.

A test case consists of a description of the starting point (also known as initial situation), the test action and the predicted result:

Starting point (initial situation)

This covers everything that is needed to prepare the system for receiving the required input. This includes not only the data that are needed for the processing, but also the condition in which the system and its environment must be. For instance, one might think of setting a specific system date, or running specific week and month batches that bring the system to a specific status.

Actions

This means all of the activities that must be executed to activate the system to the processing. It might be a simple command ('run …') or entering specific data on a screen. But it can also be a complex sequence of entering parameters, activating a specific function, manipulating other data, starting up another function, etc.

Predicted result

This covers all of the results that the tester must check to establish whether the system behavior conforms to the expectations. Often, predicted result is incorrectly thought to be limited to the output that appears on screen or is stored in databases. But the system can also produce output that is transmitted to other systems or peripheral equipment. Furthermore, more than just output data may have to be checked to establish that the system is working correctly. For instance: 'How quickly should the output appear?', 'What is the maximum allowed memory load and is it released afterwards?', or 'Should the system produce interim signals or messages, such as the hourglass or beeps?'

In other words, executing a test case roughly goes through the following steps: 'Prepare this → Do this → Check that.'

Contrary to a test situation – which addresses an isolated aspect – a test case is a complete unit that can be executed as a separate test.

When designing test cases, we first create logical test cases that are then worked out into physical test cases. Both terms are explained in further detail below.

Logical test cases

A logical test case describes, in logical terms, the circumstances in which the system behavior is examined by indicating which test situations are covered by the test case.

A logical test design contributes to reusable and maintainable testing. The logical test design provides insight into the tests to be run and distributes them over the test cases. To ensure the tests are run more efficiently, it can be useful to distribute tests for one logical test case over several physical test cases. Without a logical design it is difficult to derive from the test cases whether the coverage is sufficient. The logical test design provides insight and makes it possible to explain the choice to include or exclude certain test cases in the physical design. This makes the test design maintainable and reusable.

Finally, crating logical tests enables the tester to vary the depth and focus of the tests so that the developed tests have maximum added value. The test can cover the identified risk by using test design techniques cleverly: only those tests are run that produce information about the suitability of the test object.

Source: Test Goal – Results driven testing by Derk-Jan De Grood (2008)

Physical test cases

A physical test case is the concrete elaboration of a logical test case, with choices having been made for the values of all required the input and settings of the environmental factors.

The physical test case describes in practical terms what has to be done. The 3 basic elements of a test case are recognizable: What needs to be prepared? (Initial situation) What does the tester have to do? (Action) What is the expected result? (Predicted result).

The step from logical to physical test case is more than just inserting concrete values. It is also a step from 'theoretical' to 'practical'. In this step, the tester needs to look beyond the system specifications on which the logical test cases were based and ask himself: 'Do I now know everything I need to execute this test case in the Execution phase?'

Physical test cases generally contain a specific description of:

- The initial situation
- Everything that is needed to receive the in- and output of the system, such as:
    - Database with the required data
    - (Test) environmental parameters like the system date
    - State of the system under test
- Action
    - Actions needed to activate the system behavior, like running a batch program or entering data in a GUI
- Prediction of the result
    - Do we get the right message on the screen?
    - Do we receive the right data in the database?

Additional information that may be included:

- test case ID
- related requirement(s)
- test category
- author
- check boxes for whether the test can be or has been automated
- pass/fail
- remarks

Larger test cases may also contain prerequisite states or steps, and descriptions.

A written test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database or other common repository.

There are a lot of organizations that use neither test design techniques nor a logical test design. In these organizations, the test base is often directly converted into a physical test design. But the

tester will still implicitly use test design techniques during the conversion. Sound knowledge of the test design techniques and their principles still has added value.

3. Test case design techniques

Test design techniques can be defined as high level verification steps that are created to design a product or software that is free from all kinds of defects. Test design techniques can be derived from business scenarios and are categorized based on the type of testing. Based on the criticality of your business scenario, test design techniques are prioritized into low, medium and high. The test design techniques allow the team to execute the tests based on the risk factor. Below are some of the test design techniques that most development companies use:

Black-Box Test Design Technique

This technique is also called specification-based test design technique and uses external descriptions of the software such as technical specifications, design, requirements of the customers etc. Even if the tester doesn't have any knowledge of the code of software or internal structure, he/she can perform the test using the following popular methods:

Boundary value analysis

This is the best test design technique wherein the developer tests the input values at the boundaries. The input values are tested at the initial stages because there is a significant chance of causing errors in the functionality of the system if the input values are recognized at extreme ends. The boundary values include – minimum, maximum, error values and inside/outside boundaries.

Example:

Test a function for calculation of absolute value of an integer

Test cases:

Test requirement:		
Condition	Valid equivalence class	Invalid equivalence class
Absolute value	<0, >=0	N/A
Test scenarios:		
class x < 0, arbitrary value		x = -10
class x >= 0, arbitrary value		x = 100
classes x < 0, x >= 0, on boundary		x = 0
classes x < 0, x >= 0, below and above		x = -1, x = 1

Decision table testing

This is used to identify the condition of the test based on the decision tables that are associated with different conditions. Each and every decision corresponds to a relation, variable or predicate. Many decision tables include the symbol 'hyphen' means 'don't cares', which mean the decision tables have little influence on the actions that are being performed. The main advantage of using this technique is that it provides great confidence of the test cases.

Firstly; get to know a suitable function or subsystem that acts according to a combination of inputs or events. Taken system should be with fewer inputs or else combinations will become impossible. Always better to take maximum numbers of conditions, split them into subsets and use these subsets one at a time. After getting features that need to be combined, add them to a table showing all combinations of "Yes" and "No" for each of the feature.

Let's take an example of a finance application, where users pay money – monthly Repayment or year wise (the term of loan). If user chooses both options, the system will create a negotiation between two. So, there are two conditions of the loan amount, mention in the given below table,

TABLE 1: Blank decision table

Conditions	Step 1	Step 2	Step 3	Step 4
Repayment money has been mentioned				
Terms of loan has been mentioned				

Next, recognize all of the combinations in "Yes" and "No" (In Table 2). In each column of two conditions mention "Yes" or "No", user will get here four combinations (two to the power of the number of things to be combined). Note, if user has three things to combine, they will have eight combinations, with four things, there are 16, etc. Because of this, it's always good to take small sets of combinations at once. To keep track on combinations, give alternate "Yes" and "No" on the bottom row, put two "Yes" and then two "No" on the row above the bottom row, etc., so the top row will have all "Yes" and then all "No" (Apply the same principle to all such tables).

TABLE 2: Decision table – Input combination

Conditions	Step 1	Step 2	Step 3	Step 4
Repayment money has been mentioned	Y	Y	N	N

Terms of loan has been mentioned	Y	N	Y	N

In the next step, recognize the exact outcome for each combination (In Table 3). In this example, user can enter one or both of the two fields. Each combination is sometimes referred to as a step.

TABLE 3: Decision table – Combinations and outcomes

Conditions	Step 1	Step 2	Step 3	Step 4
Repayment money has been mentioned	Y	Y	N	N
Terms of loan has been mentioned	Y	N	Y	N
Actions/Outcomes				
Process loan money	Y	Y		
Process term	Y		Y	

At this time you didn't think that what will happen when customer don't enter anything in either of the two fields. The table has shown a combination that was not given in the specification for this example. This combination can result as an error message, so it is necessary to add another action (In Table 4). This will flash the strength this method to find out omissions and ambiguities in specifications.

TABLE 4: Decision table – Additional outcomes

Conditions	Step 1	Step 2	Step 3	Step 4
Repayment money has been mentioned	Y	Y	N	N
Terms of loan has been mentioned	Y	N	Y	N
Actions/Outcomes				
Process loan money	Y	Y		
Process term	Y		Y	
Error message				Y

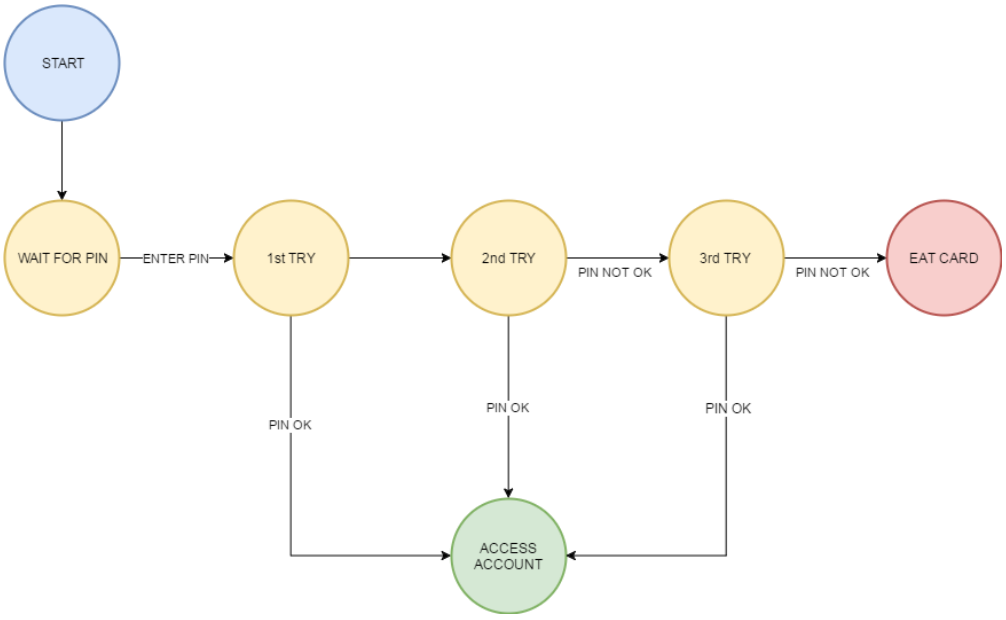
Source: http://www.softwaretestingclass.com

State transition testing

It's used to identify the conditions of the test from a state table. The state table can be considered as a truth table, wherein some inputs represent the current state, while other inputs represent the next state. This test design technique is appropriate for applications that have implemented workflow within them. Find out more about black-box testing in our course on Software Testing.

Example: Go to ATM machine to withdraw $1000, requires to the ATM machine and get cash. Later on, go to the ATM and give the same request at same amount but this time ATM refuses the amount because of insufficient balance amount.

This refusal happened just because of bank account has been changed from one state (sufficient funds) to another state (insufficient funds). The transaction that caused the account to change its state was maybe the earlier withdrawal.



Equivalence class partitioning

Equivalence partitioning (EP) is a specification-based or black-box technique. It can be applied at any level of testing and is often a good technique to use first.

- The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'. Equivalence partitions are also known as equivalence classes – the two terms mean exactly the same thing.
- In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we

assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

For example, a savings account in a bank has a different rate of interest depending on the balance in the account. In order to test the software that calculates the interest due, we can identify the ranges of balance values that earn the different rates of interest. For example, 3% rate of interest is given if the balance in the account is in the range of $0 to $100, 5% rate of interest is given if the balance in the account is in the range of $100 to $1000, and 7% rate of interest is given if the balance in the account is $1000 and above, we would initially identify three valid equivalence partitions and one invalid partition as shown below.

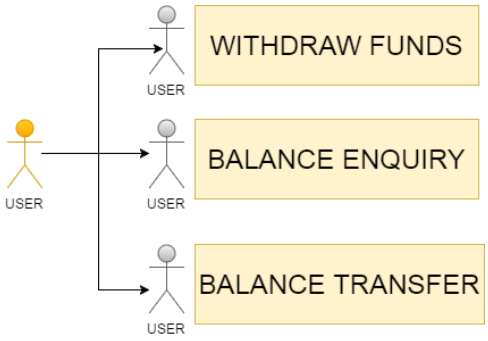
Invalid partition	Valid (3% interest)		Valid (5% interest)		Valid (7% interest)
-$0.01	$0.00	$100.00	$100.01	$999.99	$1000.00

In the above example we have identified four partitions, even though the specification mentioned only three. This shows a very important task of the tester that is a tester should not only test what is in the specification, but should also think about things that haven't been specified. In this case we have thought of the situation where the balance is less than zero.

Source: http://istqbexamcertification.com/what-is-equivalence-partitioning-in-software-testing/

Use case testing

This method is used to derive the test condition from the use case. The test conditions derived from this technique are end to end business cases. In this kind of testing, the test cases are designed to execute business scenarios and user-end functionalities.



White-Box Test Design Technique

These techniques are based on the internal structure of the program and software code, going into the minute details of the developed code and testing them one by one. Here, the tester should have proper knowledge of coding and internal structure.

Condition coverage

A major percentage of the outcomes of test conditions are exercised by this technique. Condition testing is a white box test design technique, wherein the test cases are designed in such a way that the condition outcomes are executed.

Example:

```
if ((A || B) && C)

{ << Few Statements >>}

else

{ << Few Statements >>}
```

In order to ensure complete Condition coverage criteria for the above example, A, B and C should be evaluated at least once against "true" and "false".

So, in our example, the 3 following tests would be sufficient for 100% Condition coverage testing.

A = true | B = not eval | C = false

A = false | B = true | C = true

A = false | B = false | C = not eval

Decision coverage

Here the test design technique exercises the percentage of the outcome of the decisions and designed in such a way that decision outcomes are executed.

Decision Coverage = (Number of decision outcomes executed/Total number of decision outcomes) x 100%

Statement coverage

This technique is involved in calculating the percentage of executable statements that are being exercised by the test suite.

Statement coverage = (Number of executes statements/Number of total statements) x 100

Experience Based Test Design Technique

This kind of test design technique is not involved with internal and external structure, but is based on experience. The following methods of testing are can be adopted:

Exploratory testing

Used to test the applications without any test case documentation, it is usually conducted by a business analyst and other business experts

Fault attack

This is one of the widely used techniques in experience based testing wherein the testers are allowed to anticipate the errors based on their experience, availability of the defect data and common knowledge on why a product normally fails.

Conclusion

While writing a test for a business scenario, it is advisable to keep the following points in mind:

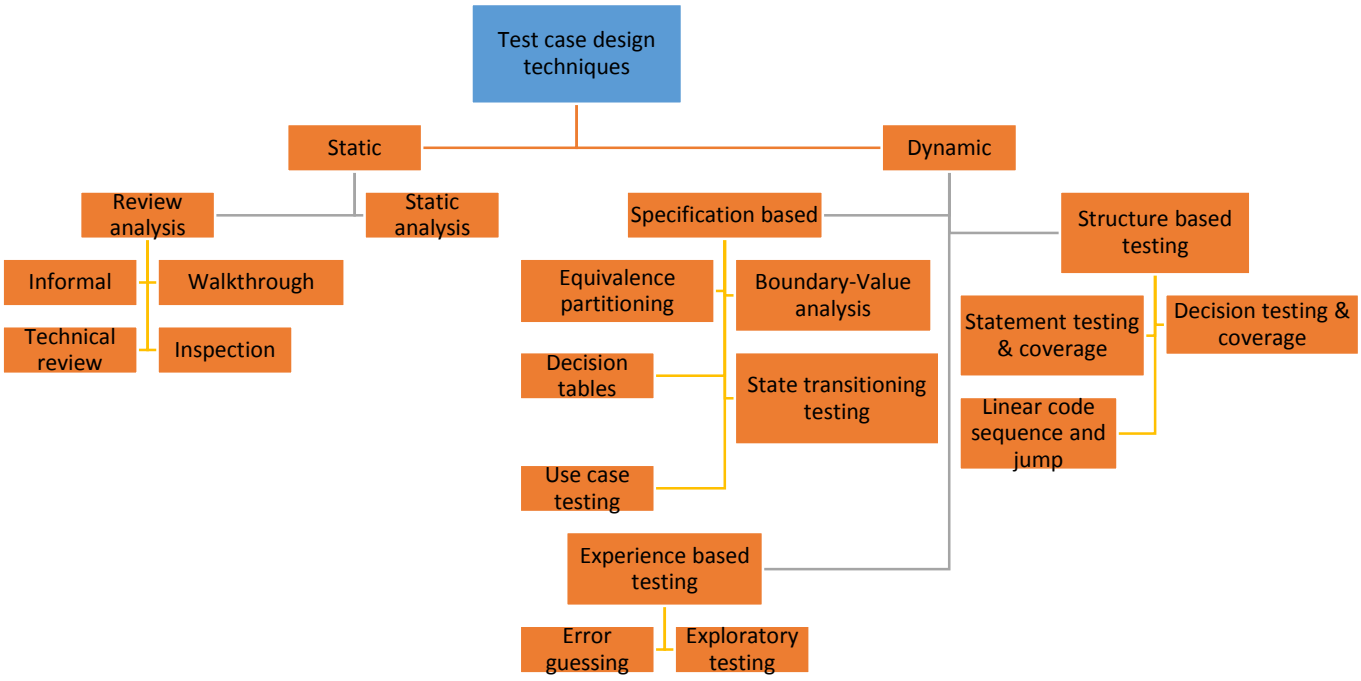
- To attain 100% logical coverage, it is advised to use decision test design technique
- If you want to cover a broad array of inputs, you can use boundary value analysis and equivalence partitioning
- For field level validations, combinations and permutations may be used

The above test design techniques help testers to select a good set of tests based on the number of possible tests for any given product. All the above test design techniques have their own strength and weakness. Each individual technique is good at finding certain kinds of defects, whereas poor at finding others. However, you must choose the best set of test design techniques to achieve maximum coverage. With the right combination of all the above techniques, you will surely be able to discover the best test scenario for all your testing requirements.

Source:

https://blog.udemy.com/test-design-techniques/

http://people.cs.aau.dk/~ask/ITV-MD1/Seminar3/Material/Slides-pdf/testcasedesign.pdf



4. Test planning

Test planning is the practice of preparing for the testing phase of product development to ensure that what is delivered to the client indeed satisfies the requirements as agreed upon in the requirement and design specification documents. The test plan helps prepare those with a stake in the tests by setting stakeholder expectations and documenting the agreed upon testing approaches.

The project test plan is a document that outlines for project stakeholders the product functions to be tested, what specific tests will be performed, the approach to be taken for those tests, what to test and what not to test, how the tests will be performed, who will be responsible for performing each test, what results are expected, what is considered a successful test and a failed test, and exit criteria for any series of tests as well as for the testing phase as a whole. A well-defined test plan must also describe, in explicit detail, the method, goals, approaches, etc. to be used for each test.

The test plan is often created in the early stage of the deployment phase by the testing team, or some group of testing specialists associated with the project. The initial draft of the plan is then reviewed by the project manager and other stakeholders to ensure its compliance with project and business requirements.

There are a number of different testing techniques and approaches. However, regardless of which approach is used test planning is made up of three basic phases that include:

Preparing for Tests

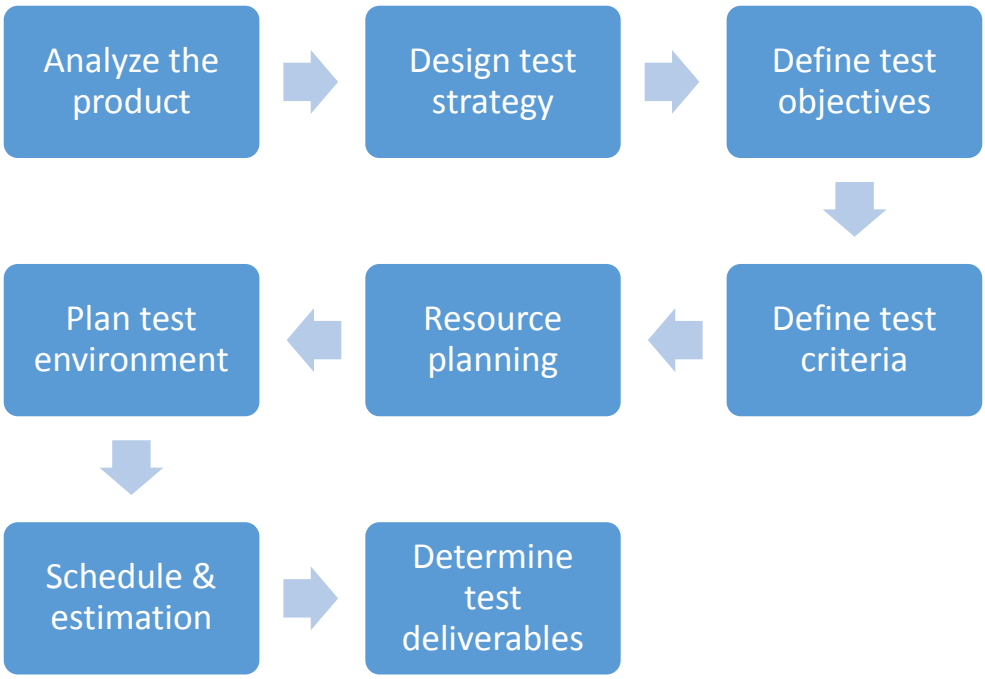
Preparation for testing is a vital part of the test planning process. This stage outlines the tests to be performed, and if necessary, creates the environment in which to perform those tests. Some of the documents necessary to effectively prepare for testing include:

Test Plan

Describes what testing will be done, to what quality standard, with what resource, within what timeframe, and outlines any risks/issues and how they will be addressed. A well-defined test plan should also outline items such as:

- Items to test and not to test such as test product features, interfaces, reporting tools
- Risks, issues, mitigation strategies, and contingencies plans
- Testing approach defining methods and tools to be used for testing
- Item pass/fail criteria specifying what constitutes a successful test
- Entry and exit criteria specifying what constitutes a completed test
- Test deliverables such as a test plan, test cases, test tools
- Environmental needs outlining any requirements for where testing will take place
- Staffing and training needs
- Acceptance criteria

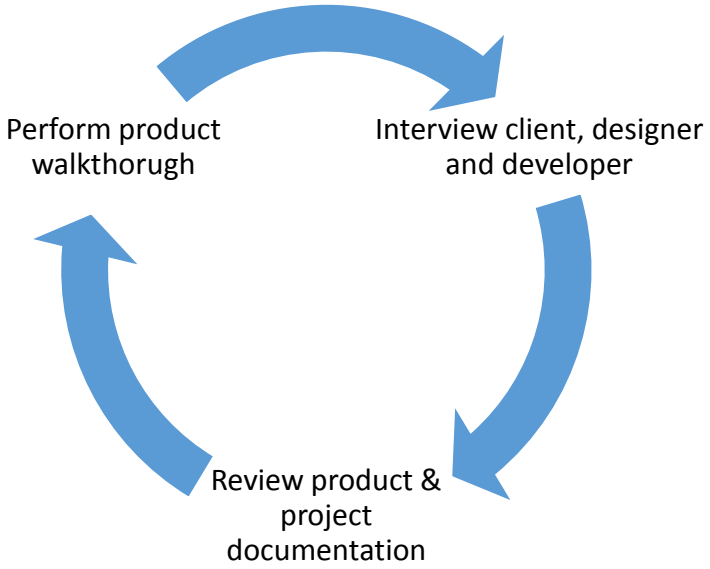
The above points fall into the different categories of the test plan creation process. This process, as represented below, should be employed for the producing of a successful test plan:



While all phases are critical for the success of the testing phase, we will only detail a couple of key steps which should be visible to the entire test team, leaving the remaining topics to the Test Lead for coverage (not in scope of this course).

Product analysis

A general approach for an efficient product analysis is described in the below workflow:



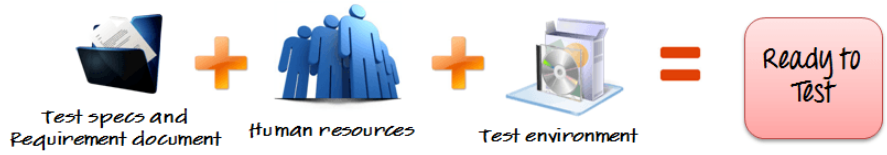
Test Strategy Design

Designing the test strategy also employs a process that will ensure that the project test objectives, effort and costs are correctly identified. This can be easily mapped to the below schema:

Definition of entry & exit criteria

Entry and exit criteria are critical for understanding when a certain test activity can be started, finished or suspended. These are usually defined as a checklist and followed up upon completion.

Defining all the pre-requisites required for testing kick-off are represented in the below diagram:



A good identification of the test suspension criteria can also be a key factor of the planning phase as this activity needs to take into account both positive and negative outcomes of the testing process. Test suspension criteria can occur when all planned tests are completed successfully (happy path) or when the stability of the system to be tested goes below a defined threshold. Resuming of test activities also happen based on a defined process:

Test deliverables

Test deliverables are critical deliverables for the software development process as they detail the system status, conformance to regulatory and stakeholder requirements as well as quality related attributes. At a high level, testing deliverables can be grouped as per the following categories:

- Before testing
  - Test plan document
  - Test cases documents
  - Test design specifications
- During testing
  - Test Scripts
  - Test Data
  - Test Traceability Matrix
  - Error logs and execution logs
- After testing
  - Test Results/Reports
  - Defect Report
  - Release Notes

Performing Tests

The schedule of what test cases will be executed when is outlined within the project's test plan which will be executed by the project's testing team. Depending on what is being tested at which point within the project's life cycle informal testing may be performed by developers, quality assurance, users, etc. However, final testing will be formally performed by individuals identified within the test plan. One essential prerequisite to testing is to at least have a completed software module to test, preferably a fully functional version of the software to be tested. Once ready, methods of testing may include:

- Compatibility Testing - tests the system, or a component of it, against existing systems, hardware, software, etc. to ensure compatibility.
- Conformance Testing - verifies the systems conformance to industry/organizational standards, Federal/organizational mandates and regulations, etc.
- Functional Testing - verifies that the system indeed conforms to documented functional specifications and ensures that the product delivered indeed satisfies the requirements as agreed upon by the client.
- Load Testing - executes performance tests and stress tests to ensure that the system can handle all expected, and exceptional, levels of demands upon the system.

- Performance Testing - executed to better understand the scalability of the system, to benchmark performance, identify performance bottlenecks, etc.
- Regression Testing - retests previously tested system components to ensure that any reported defects have been corrected and that no new quality issues have been introduced.
- Stress Testing - tests that evaluate the system, or component of it, at or beyond the limits of its specified requirements to determine the load under which it fails and how.
- System Testing - tests the entire system from end-to-end.
- Unit Testing - tests individual components or modules of the system for the purpose of finding defects.
- User Acceptance Testing - tests executed by the user to ensure that the product delivered indeed satisfies the requirements as agreed upon by the client.
- Vulnerability Assessment Testing - tests that identify, quantify, and prioritize system vulnerabilities
- Test Log/Reporting - As each test is performed it is important to document the details and results of those tests. A test log is commonly used to accomplish this. The log records information regarding what test cases have been run and the results of those tests. This log includes items such as a description of the test, the assignment of each test to one or more individuals, a target date by which the test needs to be completed, and other related information. If a test has failed, a test incident report should be filed to record the issue for future resolution.

Validating/Evaluating Testing

This simply involves evaluating the test results to verify that the product tests performed in accordance with defined requirements as agreed upon by the client. The confirmed results are then recorded using a test summary document. A test summary document records all pertinent information about the testing of the product. It would document items such as:

- An assessment of actual testing vs planned testing
- A critical assessment about the quality of the system
- The number of incidents raised and any remaining outstanding issues

Source: https://www2a.cdc.gov/cdcup/library/pmg/design/tp_description.htm