Unit 4 – Software Testing

Test Management: Test Organization Test teams, tasks and Qualifications Test Planning Quality Assurance Plan, Test Plan, Prioritization Plan, Test Exit Criteria Cost and economy Aspects Test Strategies Preventive versus Reactive Approach, Analytical versus heuristic Approach Test Activity Management, Incident Management, Configuration Management Test Progress Monitoring and Control Specialized Testing: Performance, Load, Stress & Security Testing

---

Test Management Process

Test Management :
Test Management is a process where testing activities are managed to ensure high-quality and high-end testing of software applications. This method consists of tracking, organization, controlling process, checks the visibility of the testing process in order to deliver a high-quality software application. It makes sure the software testing process run as expected.

Test Management Process :

It is a software process that manages the start to the end of all software testing activities. This management process provides planning, controlling, tracking, and monitoring facilities throughout the whole group cycle, these process includes several activities like test case design and test execution, test planning, etc. It also gives initial plan and discipline specifications to the software testing process.
Responsibilities:

- Works in collaboration with test analyst and technical test analyst to select and customizes the appropriate templates and also establish standards.
- Provides all facilities to keep track and control the testing throughout the project.
- Gives the clear concept of understanding of the testing activity of the prior upcoming project and also post ones

Test management process has **two main parts of test Management Process:**

Planning :

- Risk analysis
- Test Estimation
- Test planning

Execution :

- Testing Activity
- Issue Management
- Test report and evolution

The activity of the test process :

1. **Test plan –**
   Rough sketches are served in order to test plans to convey the process of testing. Gives a clear vision of the complete testing process.
2. **Test design –**
   Test design affords the implementation process of testing.
3. **Test execution –**
   It shows the actual system result against the expected result during test execution.
4. **Exit criteria –**
   It gives the signal when to stop the test execution process.
5. **Test reporting –**
   Test reporting picturizes the test process and result for the particular testing cycle

Tools :

Some commonly used tools are listed below-

1. qTest
2. Zephyr
3. Test Collab
4. XQual
5. TestRail
6. Testpad

Advantages:

- Reuses current test and compares the results with last trials.
- Prevents duplicate issues
- Enables conceptual graphical visualization regarding reports
- Reports errors via email

- Combines easily with automation tools and C
- Deals with tests accordingly to the requirements effortlessly
- Integrated quickly with slack
- Deals test based on cycle and sprints

Disadvantages:

- It is not cost-effective
- Doesn't support cloud-based application
- No mobile app support

Test Plan

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.
As per ISTQB definition: "Test Plan is A document describing the scope, approach, resources, and schedule of intended test activities."

What is the Importance of Test Plan?

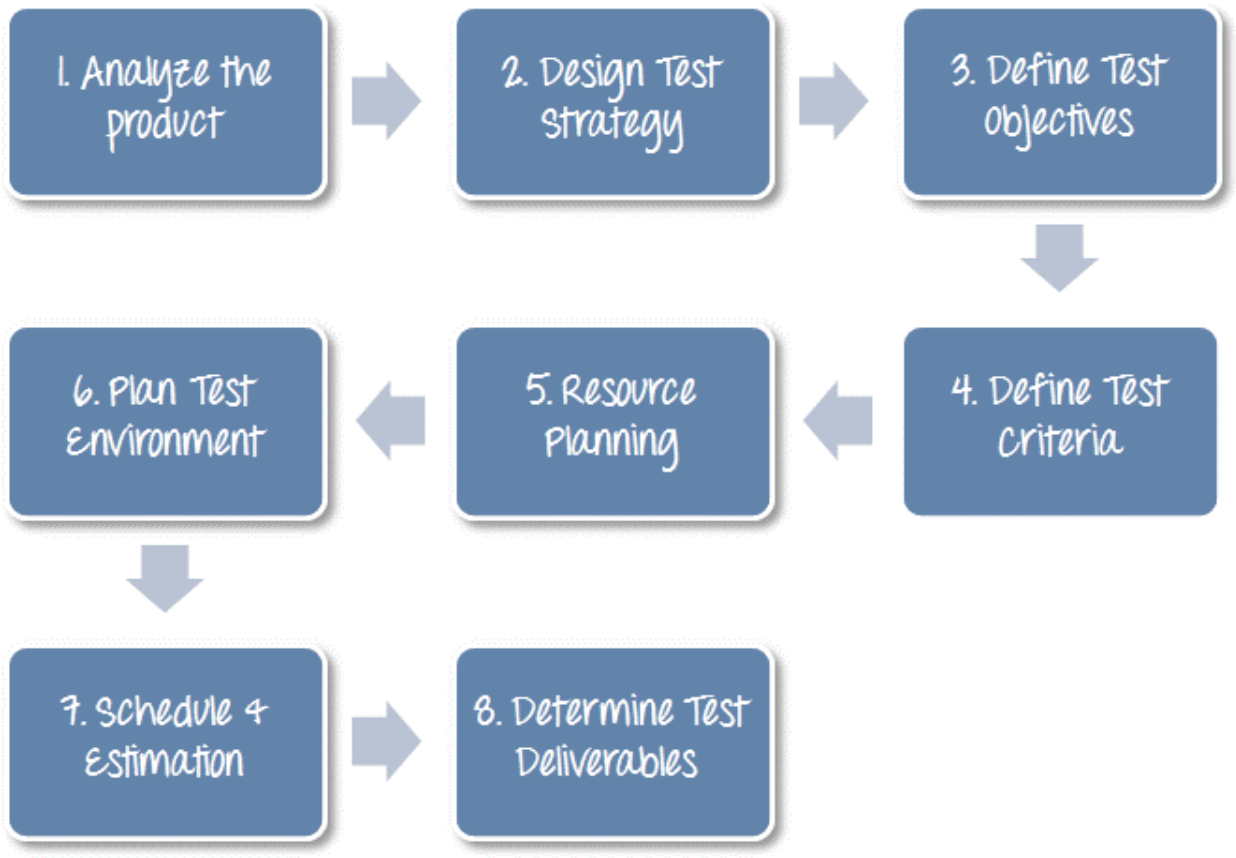
Making Test Plan document has multiple benefits

- Help people outside the test team such as developers, business managers, customers **understand** the details of testing.
- Test Plan **guides** our thinking. It is like a rule book, which needs to be followed.
- Important aspects like test estimation, test scope, Test Strategy are **documented** in Test Plan, so it can be reviewed by Management Team and re-used for other projects.

How to write a Test Plan

You already know that making a **Test Plan** is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

1. Analyze the product
2. Design the Test Strategy
3. Define the Test Objectives
4. Define Test Criteria
5. Resource Planning
6. Plan Test Environment
7. Schedule & Estimation
8. Determine Test Deliverables



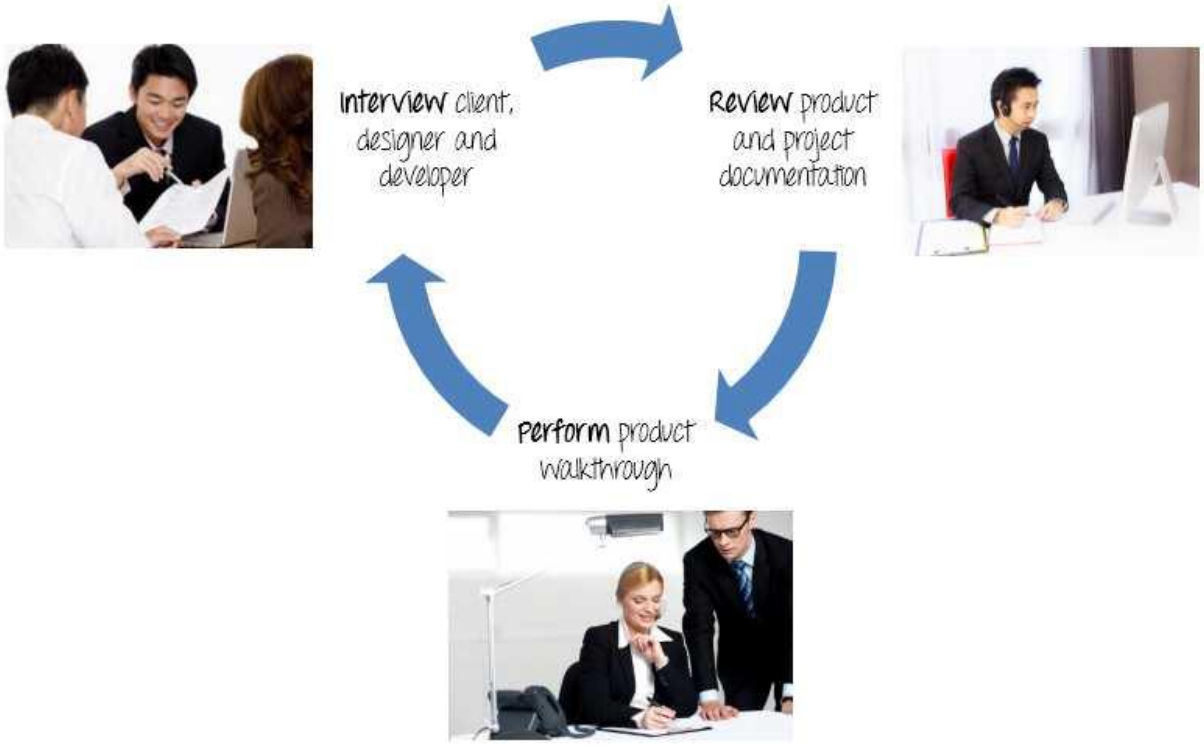
Step 1) Analyze the product

How can you test a product **without** any information about it? The answer is **Impossible.** You must learn a product **thoroughly** before testing it.

The product under test is Guru99 banking website. You should research clients and the end users to know their needs and expectations from the application

- Who will use the website?
- What is it used for?
- How will it work?
- What are software/ hardware the product uses?

You can use the following approach to analyze the site



Now let's apply above knowledge to a real product: **Analyze** the banking website `http://demo.guru99.com/V4`.

You should take a **look around** this website and also **review** product documentation. Review of product documentation helps you to understand all the features of the website as well as how to use it. If you are unclear on any items, you might **interview** customer, developer, designer to get more information.

Step 2) Develop Test Strategy

Test Strategy is a **critical step** in making a Test Plan in Software Testing. A Test Strategy document, is a high-level document, which is usually developed by Test Manager. This document defines:

- The project's **testing objectives** and the means to achieve them
- Determines testing **effort** and **costs**

Back to your project, you need to develop Test Strategy for testing that banking website. You should follow steps below

Step 2.1) Define Scope of Testing

Before the start of any test activity, scope of the testing should be known. You must think hard about it.

- The components of the system to be tested (hardware, software, middleware, etc.) are defined as "**in scope**"
- The components of the system that will not be tested also need to be clearly defined as being "**out of scope**."

Defining the scope of your testing project is very important for all stakeholders. A precise scope helps you

- Give everyone a **confidence & accurate information** of the testing you are doing
- All project members will have a **clear** understanding about what is tested and what is not

How do you determine scope your project?

To determine scope, you must –

- Precise customer requirement
- Project Budget
- Product Specification
- Skills & talent of your test team

Now should clearly define the "in scope" and "out of scope" of the testing.

- As the software requirement specs, the project Guru99 Bank only focus on testing all the **functions** and external interface of website **Guru99** Bank (**in scope** testing)
- Nonfunctional testing such as **stress**, **performance** or **logical database** currently will not be tested. (**out of** scope)

Problem Scenario

The customer wants you to test his API. But the project budget does not permit to do so. In such a case what will you do?

Well, in such case you need to convince the customer that Api Testing is extra work and will consume significant resources. Give him data supporting your facts. Tell him if Api Testing is included in-scope the budget will increase by XYZ amount.

The customer agrees and accordingly the new scopes, out of scope items are

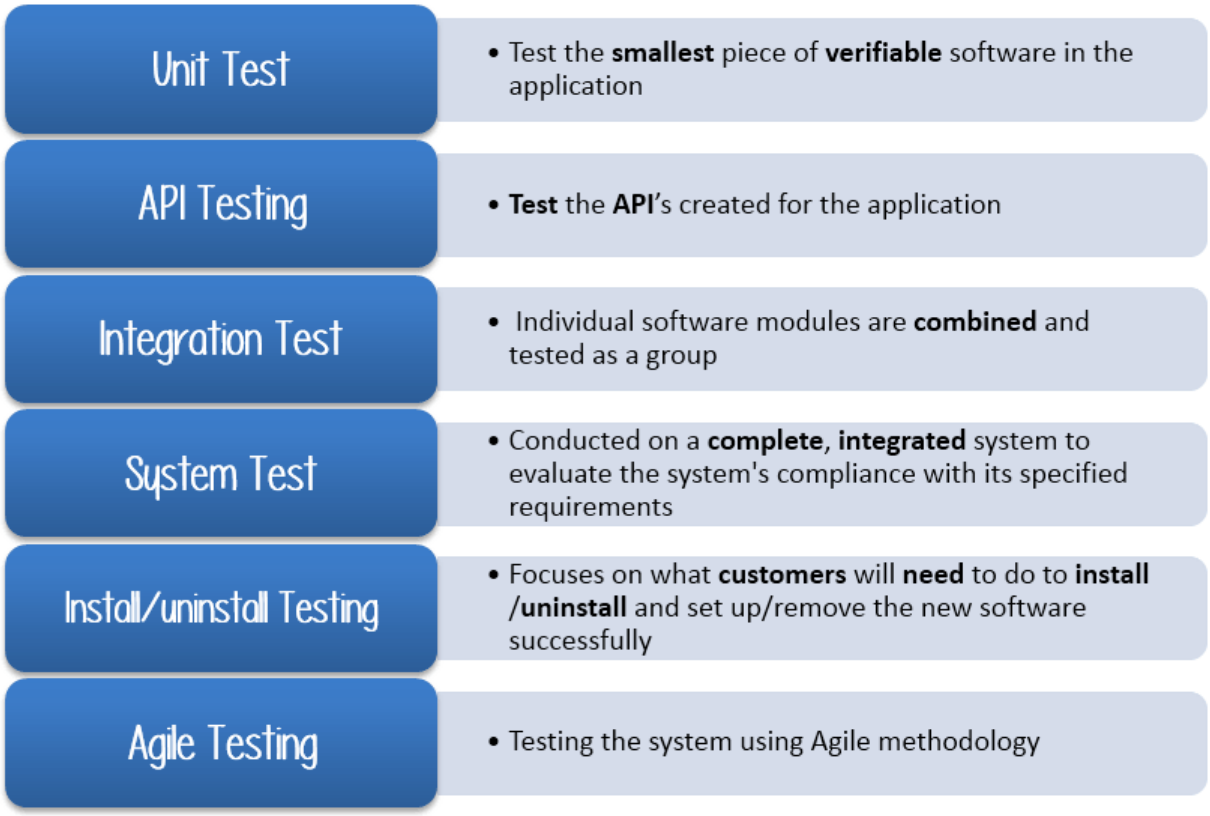
- In-scope items: Functional Testing, Api Testing
- Out of scope items: Database Testing, hardware & any other external interfaces

Step 2.2) Identify Testing Type

A **Testing Type** is a standard test procedure that gives an expected test outcome.

Each testing type is formulated to identify a specific type of product bugs. But, all Testing Types are aimed at achieving one common goal "**Early detection of** all the defects before releasing the product to the customer"

The **commonly used** testing types are described as following figure



Commonly Used Testing Types

There are **tons of Testing Types** for testing software product. Your team **cannot have** enough efforts to handle all kind of testing. As Test Manager, you must set **priority** of the Testing Types

- Which Testing Types should be **focused** for web application testing?
- Which Testing Types should be **ignored** for saving cost?

Step 2.3) Document Risk & Issues

Risk is future's **uncertain event** with a probability of **occurrence** and a **potential** for loss. When the risk actually happens, it becomes the '**issue'.**

In the article Risk Analysis and Solution, you have already learned about the 'Risk' analysis in detail and identified potential risks in the project.

In the QA Test Plan, you will document those risks

Risk	Mitigation

Team member lack the required skills for website testing.	Plan **training course** to skill up your members
The project schedule is too tight; it's hard to complete this project on time	Set **Test Priority** for each of the test activity.
Test Manager has poor management skill	Plan **leadership training** for manager
A lack of cooperation negatively affects your employees' productivity	**Encourage** each team member in his task, **and inspire** them to greater efforts.
Wrong budget estimate and cost overruns	Establish the **scope** before beginning work, pay a lot of attention to project planning and constantly track and measure the progress

Step 2.4) Create Test Logistics

In Test Logistics, the Test Manager should answer the following questions:

- **Who** will test?
- **When** will the test occur?

Who will test?

You may not know exact names of the tester who will test, but the **type of tester** can be defined.

To select the right member for specified task, you have to consider if his skill is qualified for the task or not, also estimate the project budget. Selecting wrong member for the task may cause the project to **fail** or **delay**.

Person having the following skills is most ideal for performing software testing:

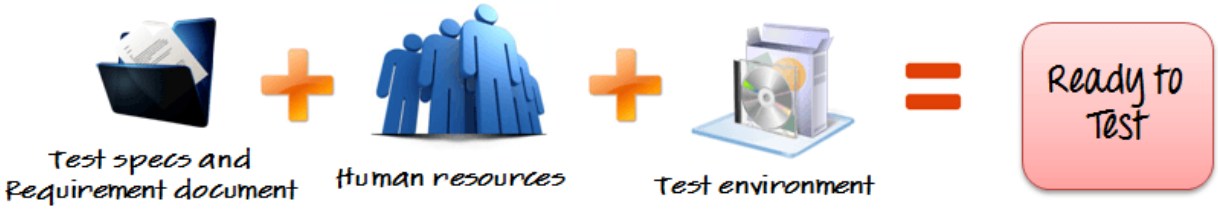
- Ability to **understand** customers point of view
- Strong **desire** for quality
- **Attention** to detail
- Good **cooperation**

In your project, the member who will take in charge for the test execution is the **tester.** Base on the project budget, you can choose in-source or outsource member as the tester.

When will the test occur?

Test activities must be matched with associated development activities.

You will start to test when you have **all required items** shown in following figure



Step 3) Define Test Objective

Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is **bug free** before release.

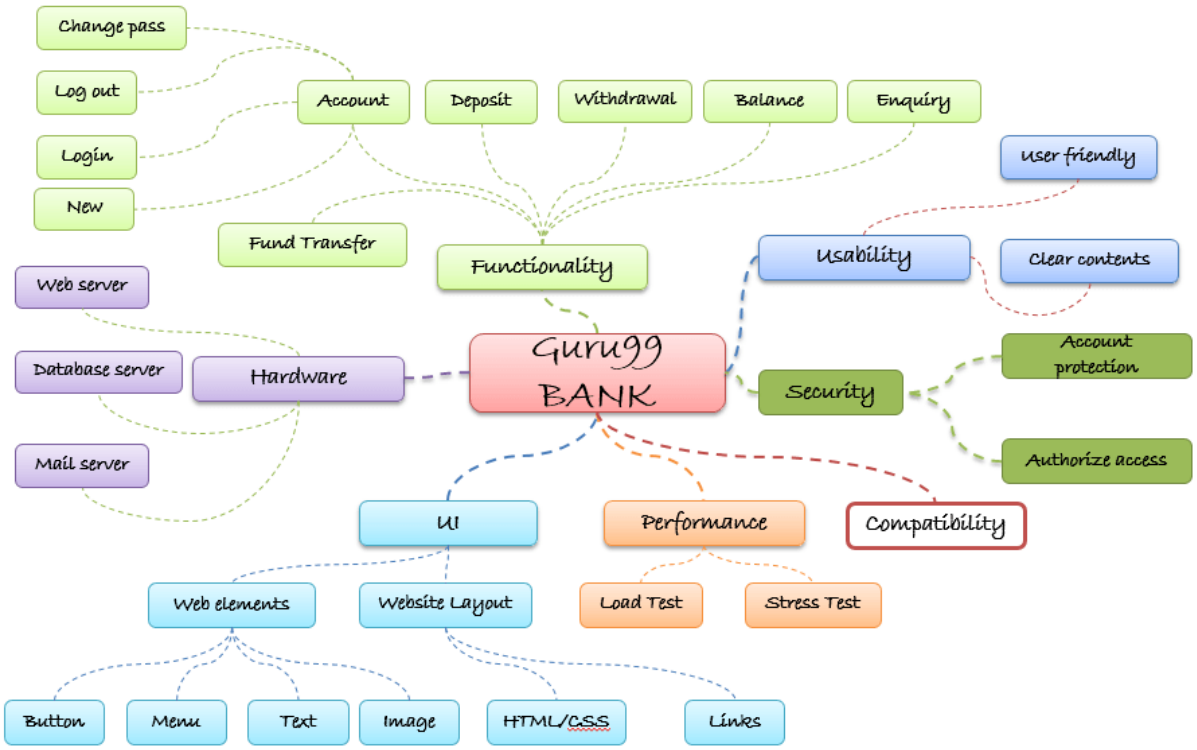
To define the test objectives, you should do 2 following steps

1. List all the software features (functionality, performance, GUI…) which may need to test.
2. Define the **target** or the **goal** of the test based on above features

Let's apply these steps to find the test objective of your Guru99 Bank testing project

You can choose the '**TOP-DOWN'** method to find the website's features which may need to test. In this method, you break down the application under test to **component** and **sub-component**.

In the previous topic, you have already analyzed the requirement specs and walk through the website, so you can create a **Mind-Map** to find the website features as following



This figure shows all the features which the Guru99 website may have.

- Check that whether website Guru99 **functionality**(Account, Deposit…) is working as expected without any error or bugs in real business environment
- Check that the external interface of the website such as **UI** is working as expected and & meet the customer need
- Verify the **usability** of the website. Are those functionalities convenient for user or not?

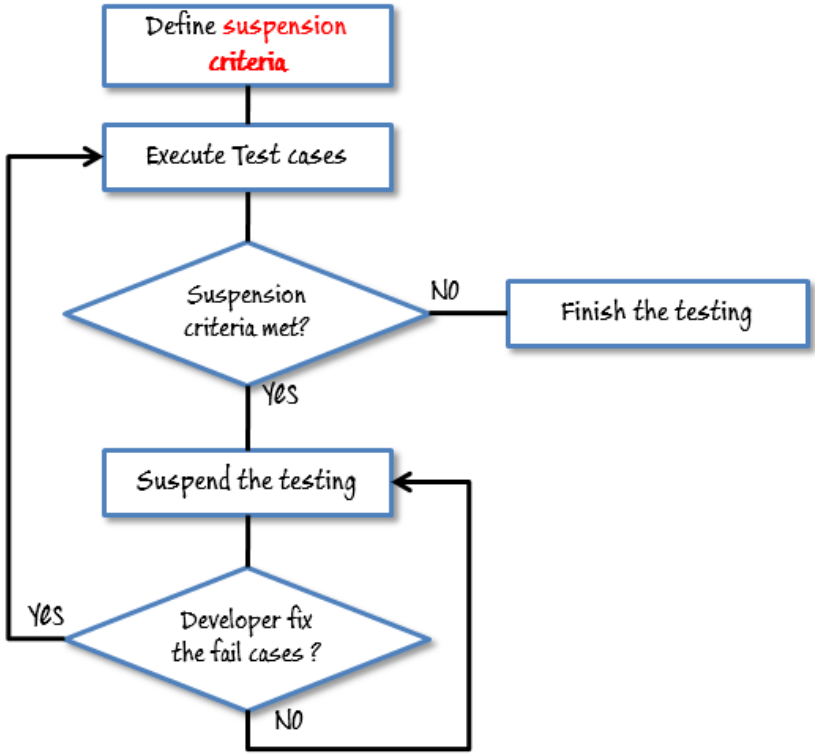
Step 4) Define Test Criteria

Test Criteria is a standard or rule on which a test procedure or test judgment can be based. There're 2 types of test criteria as following

Suspension Criteria

Specify the critical suspension criteria for a test. If the suspension criteria are met during testing, the active test cycle will be **suspended** until the criteria are **resolved**.

Test Plan Example: If your team members report that there are **40%** of test cases failed, you should **suspend** testing until the development team fixes all the failed cases.



Exit Criteria

It specifies the criteria that denote a **successful** completion of a test phase. The exit criteria are the targeted results of the test and are necessary before proceeding to the next phase of development. Example: **95%** of all critical test cases must pass.

Some methods of defining exit criteria are by specifying a targeted **run rate** and **pass rate**.

- Run rate is ratio between **number test cases executed/total test cases** of test specification. For example, the test specification has total 120 TCs, but the tester only executed 100 TCs, So the run rate is 100/120 = 0.83 (83%)
- Pass rate is ratio between **numbers test cases passed / test cases executed**. For example, in above 100 TCs executed, there're 80 TCs that passed, so the pass rate is 80/100 = 0.8 (80%)

This data can be retrieved in Test Metric documents.

- **Run** rate is mandatory to be **100%** unless a clear reason is given.
- **Pass** rate is dependent on project scope, but **achieving high pass rate** is a goal.

Test Plan Example: Your Team has already done the test executions. They report the test result to you, and they want you to confirm the **Exit Criteria.**

In above case, the Run rate is mandatory is **100%,** but the test team only completed 90% of test cases. It means the Run rate is not satisfied, so do NOT confirm the Exit Criteria

Step 5) Resource Planning

Resource plan is a **detailed summary** of all types of resources required to complete project task. Resource could be human, equipment and materials needed to complete a project

The resource planning is important factor of the test planning because helps in **determining** the **number** of resources (employee, equipment…) to be used for the project. Therefore, the Test Manager can make the correct schedule & estimation for the project.

This section represents the recommended resources for your project.

Human Resource

The following table represents various members in your project team

No.	Member	Tasks
1.	Test Manager	**Manage** the whole project Define project **directions** Acquire appropriate resources
2.	Tester	Identifying and describing appropriate test techniques/tools/automation architecture Verify and assess the Test Approach **Execute** the tests, **Log** results, **Report** the defects. Tester could be in-sourced or out-sourced members, base on the project budget For the task which required **low** skill, I recommend you choose **outsourced** members to **save** project cost.
3.	Developer in Test	**Implement** the test cases, test program, test suite etc.
4.	Test Administrator	Builds up and ensures Test Environment and assets are **managed** and **maintained** **Support**Tester to use the test environment for test execution
5.	SQA members	Take in charge of quality assurance Check to confirm whether the testing process is meeting specified requirements

System Resource

For testing, a web application, you should plan the resources as following tables:

No.	Resources	Descriptions

1.	Server	Install the web application under test This includes a separate web server, database server, and application server if applicable
2.	Test tool	The testing tool is to automate the testing, simulate the user operation, generate the test results There are tons of test tools you can use for this project such as Selenium, QTP…etc.
3.	Network	You need a Network include LAN and Internet to simulate the real business and user environment
4.	Computer	The PC which users often use to connect the web server

Step 6) Plan Test Environment

What is the Test Environment?

A testing environment is a setup of software and hardware on which the testing team is going to execute test cases. The test environment consists of **real business** and **user** environment, as well as physical environments, such as server, front end running environment.

How to setup the Test Environment

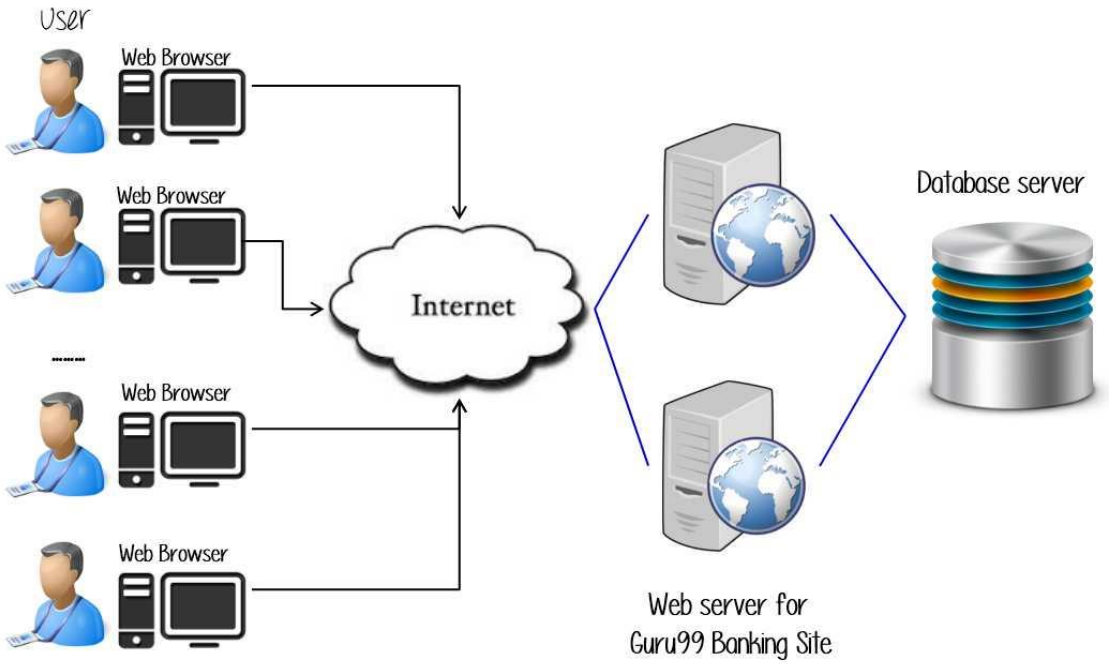
Back to your project, how do you set up **test environment** for this banking website?

To finish this task, you need **a strong cooperation** between Test Team and Development Team

You should ask the developer some questions to understand the web application under test **clearly**. Here're some recommended questions. Of course, you can ask the other questions if you need.

- What is the maximum user connection which this website can handle at the same time?
- What are hardware/software requirements to install this website?
- Does the user's computer need any particular setting to browse the website?

Following figure describes the test environment of the banking website



Step 7) Schedule & Estimation

In the article Test estimation, you already used some techniques to estimate the effort to complete the project. Now you should include that estimation as well as the schedule to the Test Planning

In the Test Estimation phase, suppose you break out the whole project into small tasks and add the estimation for each task as below

Task	Members	Estimate effort
Create the test specification	Test Designer	170 man-hour
Perform Test Execution	Tester, Test Administrator	80 man-hour
Test Report	Tester	10 man-hour
Test Delivery		20 man-hour
Total		**280 man-hour**

Then you create the **schedule** to complete these tasks.

Making schedule is a common term in project management. By creating a solid schedule in the Test Planning, the Test Manager can use it as tool for monitoring the project progress, control the cost overruns.

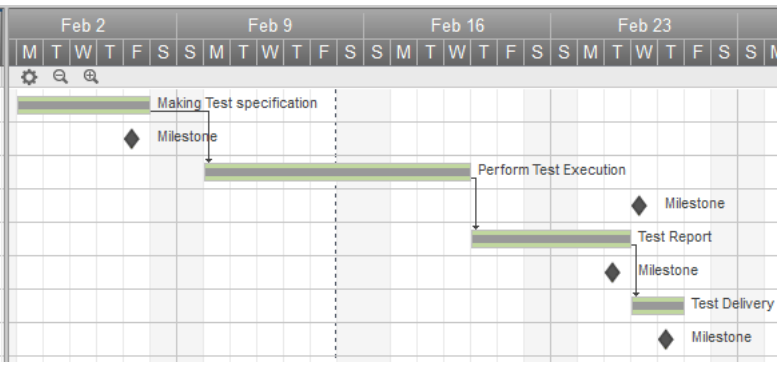
To create the project schedule, the Test Manager needs several types of input as below:

- **Employee and project deadline**: The working days, the project deadline, resource availability are the factors which affected to the schedule
- **Project estimation**: Base on the estimation, the Test Manager knows how long it takes to complete the project. So he can make the appropriate project schedule
- **Project Risk** : Understanding the risk helps Test Manager add enough extra time to the project schedule to deal with the risks

Let's practice with an example:

Suppose the boss wants to complete the project Guru99 in **one** month, you already estimated the effort for each tasks in Test Estimation. You can create the schedule as below

Task Name	Start Date	End Date
Making Test specification	**02/03/14**	**02/07/14**
Milestone	02/07/14	02/07/14
Perform Test Execution	**02/10/14**	**02/19/14**
Milestone	02/26/14	02/26/14
Test Report	**02/20/14**	**02/25/14**
Milestone	02/25/14	02/25/14
Test Delivery	**02/26/14**	**02/27/14**
Milestone	02/27/14	02/27/14



Step 8) Test Deliverables

Test Deliverables is a list of all the documents, tools and other components that has to be developed and maintained in support of the testing effort.

There are different test deliverables at every phase of the software development lifecycle.

Test deliverables are provided **before** testing phase.

- Test plans document.
- Test cases documents
- Test Design specifications.

Test deliverables are provided **during** the testing

- Test Scripts
- Simulators.
- Test Data
- Test Traceability Matrix
- Error logs and execution logs.

Test deliverables are provided **after** the testing cycles is over.

- **Test Results/reports**
- Defect Report

- Installation/ Test procedures guidelines
- **Release notes**

What is an Exit Criterion?

Exit criterion is used to determine whether a given test activity has been completed or NOT. Exit criteria can be defined for all of the test activities right from planning, specification and execution.

Exit criterion should be part of test plan and decided in the planning stage.

Examples of Exit Criteria:

- Verify if All tests planned have been run.
- Verify if the level of requirement coverage has been met.
- Verify if there are NO Critical or high severity defects that are left outstanding.
- Verify if all high risk areas are completely tested.
- Verify if software development activities are completed within the projected cost.
- Verify if software development activities are completed within the projected timelines.

Cost and economy Aspects Test Strategies

There is cost of activity in every project, it should have business value and software testing is no exception. To optimize its business value, test managers should optimize testing appropriately. There should not be unnecessary testing otherwise it causes unnecessary delays and ends up incurring more costs. Also, there should not be incomplete or too less testing otherwise, there may be a chance of defective products to be handed to the end users. Therefore, software testing should be done appropriately.

Cost of Quality :

It is the most established, effective measure of quantifying and calculating the business value of testing. There are four categories to measure cost of quality: Prevention costs, Detection costs, Internal failure costs, and External failure costs.

These are explained as follows below.

1. **Prevention costs** include cost of training developers on writing secure and easily maintainable code
2. **Detection costs** include the cost of creating test cases, setting up testing environments, revisiting testing requirements.
3. **Internal failure costs** include costs incurred in fixing defects just before delivery.
4. **External failure costs** include product support costs incurred by delivering poor quality software.

Major parts of total cost are detecting defects and internal failure cost. But, these costs less than external failure costs. That's why testing provides good business value.

Business value of software testing :

Test manager has responsibility to identify the business value and provide communications between teams and senior management. It concerns the cost of quality.

To deliver business value, these are some of the measurable ways, like

1. Detecting defects
2. Documenting defects
3. Status reports and test metrics on project progress
4. Status reports on process implementation and product development
5. Increasing confidence in product quality
6. Legal liabilities

7. Decreasing risks
8. Ensuring predictable product
9. Enhancing reputation for product
10. Enhancing reputation for process quality\
11. Testing can prevent mission failure

Differences between Black Box Testing vs White Box Testing

1. **Black Box Testing** is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. Only the external design and structure are tested.

2. **White Box Testing** is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Implementation and impact of the code are tested.

Differences between Black Box Testing vs White Box Testing:

S. No.	Black Box Testing	White Box Testing
1.	It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.

8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
9.	It is the behavior testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.
12.	It is least time consuming.	It is most time consuming.
13.	It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
14.	Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
15.	**Example:** Search something on google by using keywords	**Example:** By input to check and verify loops
16.	**Black-box test design techniques-** • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	**White-box test design techniques-** • Control flow testing • Data flow testing • Branch testing
17.	**Types of Black Box Testing:** • Functional Testing • Non-functional testing • Regression Testing	**Types of White Box Testing:** • Path Testing • Loop Testing • Condition testing
18.	It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

What is Test Approach? What is the difference between Preventative Approach and Reactive Approach?

Implementation of test strategy for a particular project is known as "test approach". The test approach is usually defined in all test plans and test designs. Test approach refers to the commencement of various project activities such as planning the testing process, selecting the designs, defining the entry and exit criteria etc.

There are two approaches in testing:

1. Preventative approach.
2. Reactive approach.

Preventative approach

In preventative approach, tests are designed at an early stage, i.e. before the commencement of software development.

Reactive approach

In Reactive approach, tests are designed after software development.

Before selecting a 'test approach', we should consider the following factors:

- Analyse the various risks associated with failure of the project - to the people,the environment, and the company.
- Analyse the expertise and efficiency of the people in the proposed tools and techniques.
- The nature of product and the business.

Measuring how much of your code you are testing

Fully tested source code is a measure of the quantity of source code within your application that is actually being tested. A deeper definition would be a measure of the quantity of complex business logic within your source code that is actually being tested.

The higher the percentage of code being tested is directly proportional to how proactive you are in fully testing your source code. Thus, the likelihood or statistical chance of a bug entering your source code and making it to your production environment when new code is created or current code is updated is the direct inverse of the percentage of code being fully tested; this is the proactive to reactive ratio.

For example, if only 10% of your code is being tested in a proactive way, then there is a 90% chance or likelihood that the rest of your source code will have minor to major bugs causing a disproportionate reaction. A disproportionate amount of time and resources will thus be given to:

- Discovering where the bugs exist
- Determining if and how much data was corrupted
- Writing code to make data corrections
- Modifying code to make business logic corrections

However, if you’re being proactive, you would be writing white box tests to further avoid being reactive to more bug fixes in your application’s source code in the future.

How to fully test your application's source code

So how do you fully test your application's source code? You fully test your application's source code through white box testing. And what is white box testing? **White box testing** is a method of **software testing** that examines the internal structures or workings of an application, as opposed to its functionality (**Wikipedia**). Integration tests are a good example of white box testing.

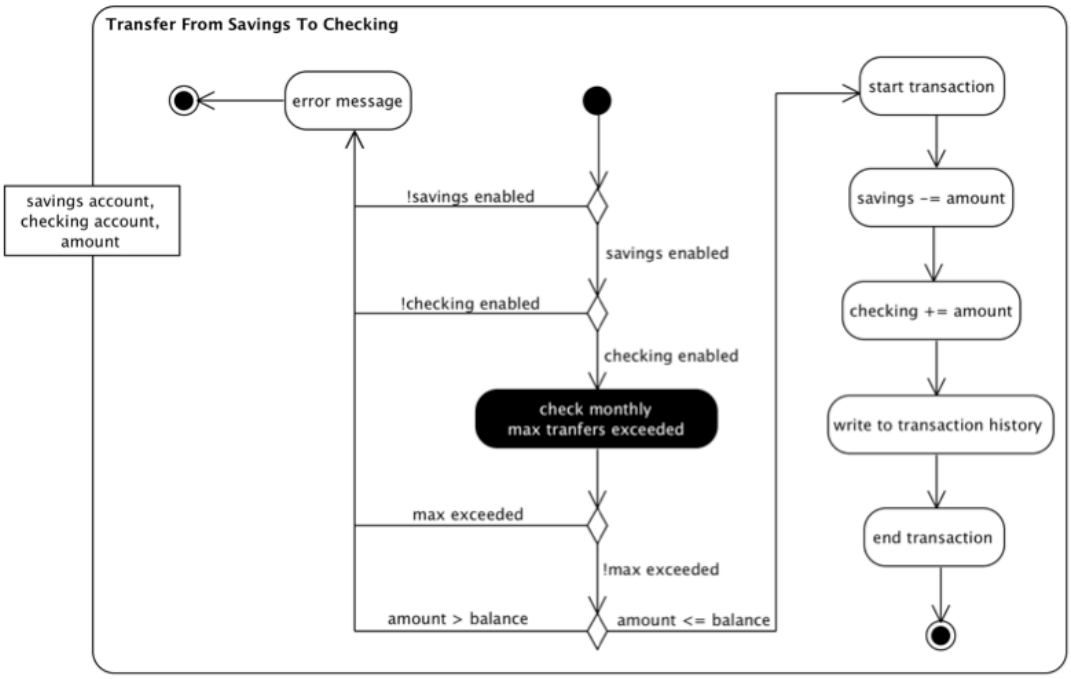
Why black box testing is not enough

So why not use black box testing? Isn't it good enough? Not really… **Black box testing** is a method of software testing that examines the functionality of an application without peering into its internal structures or workings (**Wikipedia**). Functional tests are a good example of black box testing.

You can't fully test your source code if you're using black box testing only; you'll be lucky if you can fully test 10% of your source code. Fully testing your application's source code can only come from true white box testing.

Banking application example

Take a look at the following activity diagram that describes the business logic of executing a bank transfer from a savings account to a checking account; how many tests do you see?



Let’s count them:

Test 1

- Assert savings not enabled
- Assert checking enabled

Test 2

- Assert savings enabled
- Assert checking not enabled

Test 3

- Assert savings enabled
- Assert checking enabled
- Assert monthly max transfers exceeded

Test 4

- Assert savings enabled
- Assert checking enabled

- Assert monthly max transfers not exceeded

Test 5

- Assert savings enabled
- Assert checking enabled
- Assert monthly max transfers not exceeded
- Assert amount is less than balance

Test 6

- Assert savings enabled
- Assert checking enabled
- Assert monthly max transfers not exceeded
- Assert amount is greater or equal to balance
- Assert savings is debited the value of amount
- Assert checking is credited the value of amount
- Assert a transaction history record was created
- Assert transaction history record attributes contain the correct information

For the business logic defined within this activity diagram, **a minimum of six white box tests will be necessary** to fully test transferring money from a savings account to a checking account.

Couldn’t I fully test that application with just black box testing?

Could you test all six of these conditions via black box testing, most likely not.

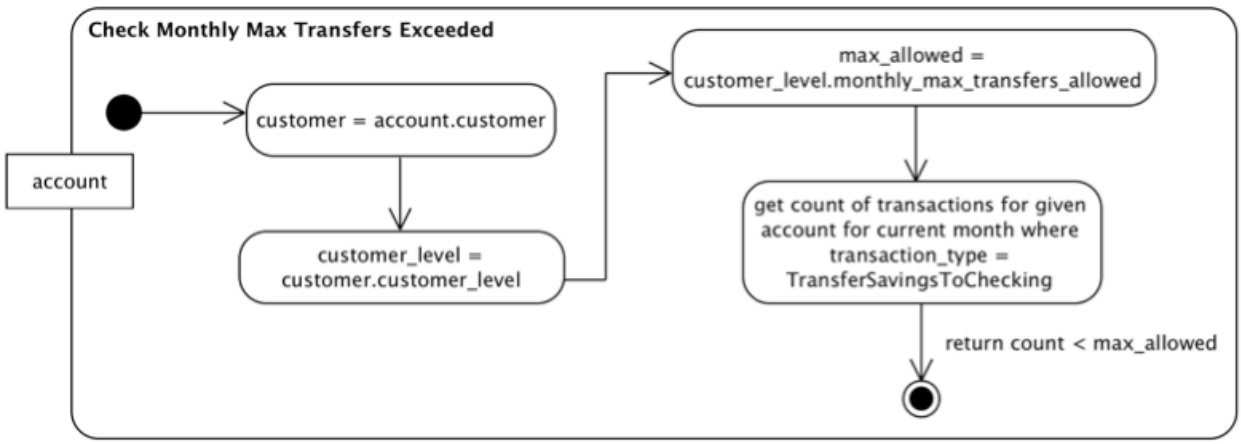
1. You’ll most likely not have an activity diagram that gives you a view into methods business logic.
2. Even if you had an activity diagram that gives you a view into the methods business logic, it’s not reliable because you can’t be sure that other business logic exists within the method that needs to be tested without physically looking at the code.

3. Through black box testing, it would be very difficult and laborious to setup the conditions for each test; at best one would setup ideal conditions to black box the last test, test number six.

I must be fully testing with those 6 white box tests then…right?

Even if you were able to peer into the source code and write the six white box tests necessary to fully test the transfer from savings to checking business logic, is the code now fully tested? No, not completely.

In the middle of the activity diagram, there is another black-box, checking monthly max transfers exceeded, that can only truly be tested via white box testing. Do you know what the code does? Let's take a look.



And until this black box of business logic is fully white box tested, then the opportunity for a savings to checking transfer to perform incorrectly is reactively high.

Let's say that initially, when this code went to production, the business logic that detects when max transfers are exceeded works perfectly; and then a minor addition to the business logic is made, but the code is not fully white box tested and thus a small but fatal bug is accidentally introduced. If the fatally bugged source code is introduced into the production environment, the fallout could be catastrophic.

The danger of not fully testing

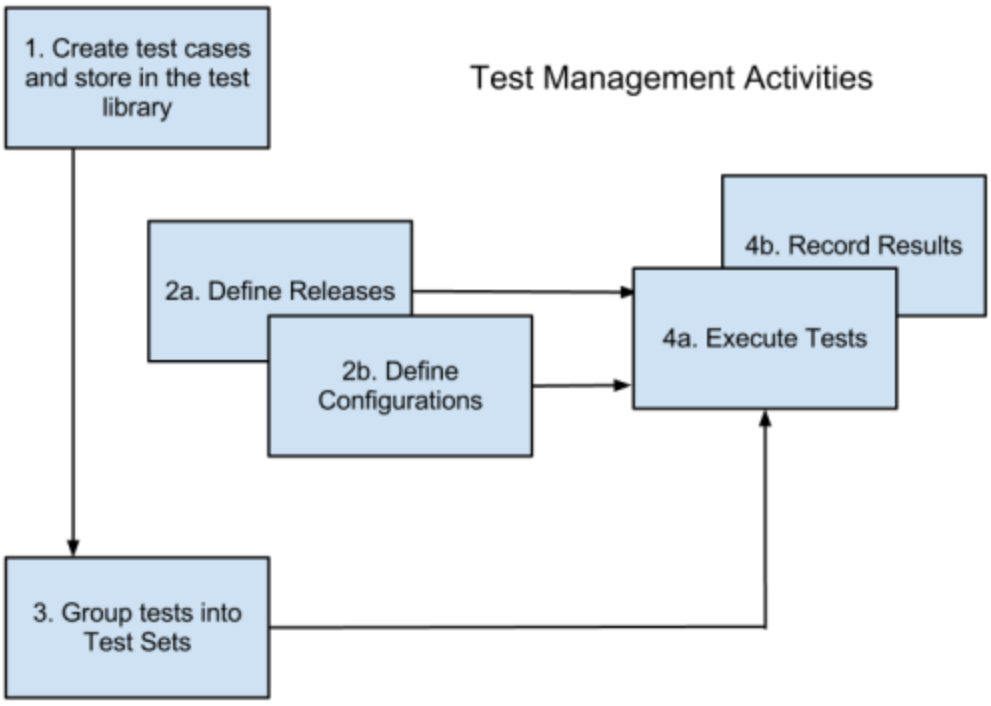
If the banking application's source code was servicing a base of 10,000,000 users and averaging 10,000 savings to checking transfers every 60 seconds, even if the bug were to exist in the production environment for as little as one hour, it could potentially affect over 600,000 users and millions of rows of data in the production database. The amount of time and number of resources that would have to react to correcting such a catastrophe would be extensive and extremely costly.

Does this sound scary? It should scare you a little or maybe a lot. If you're not proactively white box testing your application's source code, it's only a matter time before you will have to react disproportionally to the bugs that will inevitably get introduced into your production environment, wreaking havoc, costing you and your customers a lot of unnecessary pain.

This emphasizes the point of doing proactive vs reactive testing and potentially using a solution like **GenRocket** to accomplish full test code coverage.

Test Management Activity

There are a number of core test management activities that you will be performing as part of any QA process. What we cover here is good software testing practice and should form the bedrock of any QA teams setup. The activities we'll look at are :



Test Management Activities

1. Developing testcases in a library.
2. Defining releases and configurations that we'll use during the execution.
3. Grouping into sets ready for execution.
4. Execution.

In this discussion we're going to look specifically at carrying out those activities with QAComplete. With QAComplete we'll see how we can streamline this approach. We'll see how this can improve on the process that we'd normally see with an excel based approach.

The first part of this process is developing the tests. We're focusing on the test management activity of writing good testcases. At this point we're not thinking at all about the execution phase. It's important to keep these two test management activities completely separate. If we don't the temptation is to write what we see whilst we are using the application. So the case content is driven by what the application does. This of course defeats the whole point of testing. We're defining what we expect to see before we check that the application conforms to our expectations.

The second part is to define the releases and configurations that we expect to run our testcases against. There is no point running any checks if you are not recording or tracking what you are running against. The first aspects we need to track is the version, build or iteration of the product. The second aspect is the configuration of the system or

software (e.g. the operating system we're running the product on). So the test management activity we need to concern ourselves with here is identifying the release and configuration combinations we need to work with.

Once we've written our testcases and defined releases/configurations we should look at grouping our tests in to sets. This helps on several levels. Firstly a group of similar testcases is easier to manage and simpler to maintain. At execution time a group of similar tests can be quicker to run. Whilst there are no hard and fast rules on how to go about grouping the following grouping approaches could be considered:

i. type of test (e.g. integration, system, etc)
ii. feature or module
iii. process
iv. scenario
v. complexity

Clearly there are many ways to group here. However, the most important point to consider is how you would like to see your reports structured once you are finished. If you need to report on progress based on product features then grouping by feature may be a good approach.

With sets created we can then move on to the execution. At run time the test management activities we'll be focusing on will be selecting the set to run and specifying which configuration/release we'll run the set against. Once we've identified these attributes we can step though each testcase and record the results.

In principal these 4 core test management activities are quite simple to grasp. As always though the devil is in the detail. Developing and writing the testcases can be considered an art form and requires a good degree of skill. Identifying releases and configurations, whilst easy enough, is complicated massively by the combinations and permutations. Again skill and experience come in to play here. Grouping into sets is probably the easiest of the tasks here. Just keep in mind your reporting requirements. And then finally we come on to the execution. If we've completed the other tasks well this should be pretty straightforward. With all of this keep in mind that these core test management activities will remain constant and will provide a good framework to help you manage the underlying complexities.

What is Incident

Incidents can be defined in simple words as an event encountered during testing that requires review.

While testing if the actual result varies from the expected result, it is referred to as a bug, defect, error, problem, fault, or incident. Most often, all of these terms are synonymous.

Incidents however are a special category of issue that might occur due to misconfiguration, corrupted data, server crashes, etc. Examples are Disk spaces full, error in execution (Runtime Error), service unavailable, etc.

Incidents can also occur due to issues with software development, hardware usage, or service request errors.

Difference Between Errors, Defects, Bugs, and Incidents

1. **Error:** An action performed by a human that results in unexpected system behavior. E.g. incorrect syntax, improper calculation of values, misunderstanding of software requirements, etc.
2. **Defect:** This is a term usually used by testers. If the tester finds a mistake or problem then it is referred to as Defect.
3. **Bug:** Bug is the developer's terminology. Once a defect found by a tester is accepted by the developer it is called a bug. The process of rectifying all bugs in the system is called Bug-Fixing.
4. **Incident:** Incident is an unplanned interruption. When the operational status of any activity turns from working to failed and causes the system to behave in an unplanned manner, it is an incident. A problem can cause more than one incident which is to be resolved, preferably as soon as possible.

Now, let's look at a few related terms:

1. **Incident Repository:** Incident Repository can be defined as a database that contains all the important and relevant data about all incidents occurring in the system. This information was subsequently used to create the incident report. It contains fields such as data, expected results, actual results, date and time, the status of the incident, etc.
2. **Severity:** The potential impact of the incident will decide its severity. This can be Major, Minor, Fatal, or Critical for immediate resolution.
3. **Priority:** Set according to severity and influence on the working status of the system. Values can be High, Medium, Low, Very High, or Urgent/Immediate.
4. **Incident Status**: The current state where handling the incident is. This can be New, In Progress, Resolved, and Closed.

What is Incident Management?

Incident management is a process for logging, recording, and resolving incidents as quickly as possible to restore business processes or services back to normal.

Recommended Incident Management Software – Salesforce

There is hardly a better tool out there that does incident tracking and management as well as Salesforce. The platform is capable of proactively seeking out a problem and presenting agents with the ability to resolve it before the issue worsens. The fact that it can integrate with platforms like Slack also make incident handling and escalation quicker and considerably more efficient.

Features:

- Proactive Incident Detection.
- Streamline operations with real-time collaboration.
- Keep customers updated via multiple digital channels.
- Automate business processes using AI.
- Integrate with external systems for quick problem resolution.

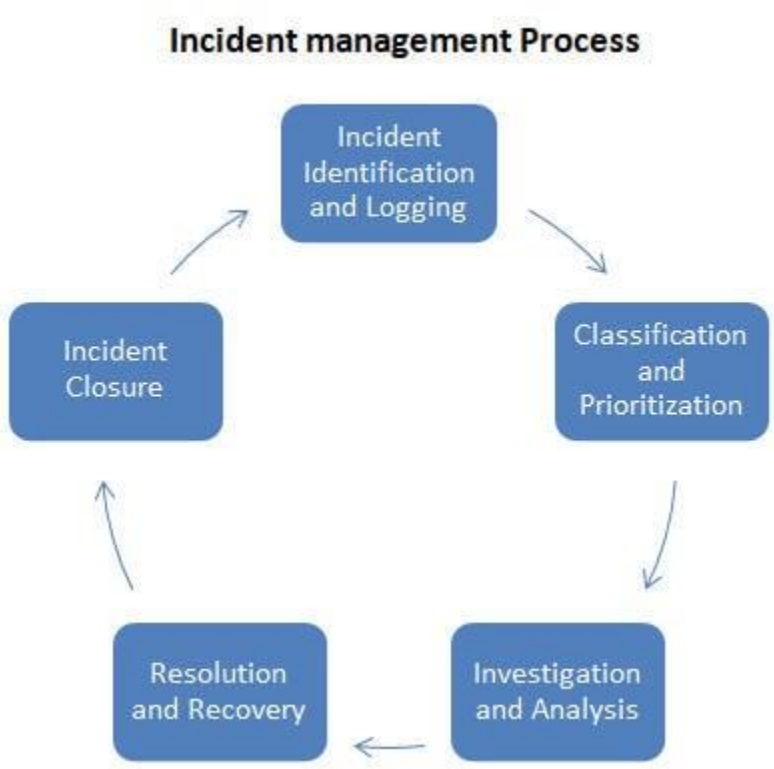
=> Visit Salesforce Website

Incident Management Process

Incident management is the overall process starting from logging incidents to resolving them.

This is a very critical process as this will ensure that the incidents get addressed in a systematic and effective manner. Also, by streamlining the entire process, there is a good chance that early fixing of the issues might happen.

The following is a diagrammatic representation of the process and we will discuss each stage in detail next.



#1) Incident Identification and Logging: Incident Identification is either done via testing (using tools or otherwise), user feedback, infrastructure monitoring, etc.

Logging an incident simply means recording the following info:

- Exact/Appropriate date and time of occurrence.
- Incident title along with the type and a brief description.
- Name of the person who logged the incident and a more detailed description. with an error code when applicable.
- Details of the person assigned to the incident for follow-up.
- Current Status of the incident
- Attachments include technical discussions, decisions, and approvals.

#2) Classification and Prioritization: Classification of incidents helps us partition them based on their type (software, hardware, service request, etc.) so it makes for easier reporting and analysis. Prioritisation helps to identify the order/priority of incidents to be handled. It depends on the impact, severity, and most importantly the Risk Factor.

#3) Investigation and Analysis: This step is to better understand the problem so we not only fix it right now but gather information to prevent it from re-occurrence.

#4) Resolution and Recovery: Steps are taken to remove the incident and bring the system back to its previous working condition.

#5) Incident Closure: The resolution is retested and in case the system is working as intended, the incident is closed.

Incident Management System

Incident management can very well be done manually or statically using spreadsheets but it is much more effective, dynamic, and systematic when done via a tool.

The incident management system is used by many Customer Support call centers to create updates and resolve incidents.

Popular Incident Management Tools:
Some popular incident management tools that can be used for tracking incidents in addition to bugs or defects are:

#1) SiT! (Support Incident Tracking)

- Support Incident Tracker (SiT) is a Free Open Source and web-based application which uses PHP and MySQL for and supports all platforms. It is also commonly known as a 'Help Desk' or 'Support Ticket System'.
- Useful for sending emails directly from SiT, attach files and record every communication in the incident log. SiT is aware of Service Level Agreements and incidents are flagged if they lie outside of them.

#2) JIRA

JIRA is also a popular proprietary incident management tool developed by Atlassian and is used for bug, defect or incident tracking. It is a Java-based tool used for software and mobile apps. JIRA scheme involves workflows, permissions, configurations, issue types, etc. JIRA also supports agile testing.
For more information and tutorials, please check: JIRA tutorials series.

#3) Incident Tracking System
Incident tracking System is software used for tracking incidents. It helps to determine and analyze the root cause of incident along with suitable solutions. Incident Tracking System is easy to use and provides database support for tracking and recording the Incident.

Test Incident Report

1. Test incident report is an entry created in the defect repository with a unique ID for each incident encountered. The test incident report documents all issues found during the various phases of testing.
2. IEEE 829-1998 is the standard format for test incident reports which are used to document each incident that occurs during testing.

The outline of the IEEE 829-1998 template is given below:
=> Download IEEE Incident Tracking Template here.
The following is a brief explanation of the fields:

#1) Identifier: Specifies ID which is a unique and company-generated number to identify and locate an incident.
#2) Summary: Summarizes the incident in a concise way. Contains sufficient details to understand related facts viz. references, associated test procedures, version of the software, test cases, etc.
#3) Incident Description: Describes incident with the following details: Inputs

- Expected Result
- Actual Result
- Attempt to repeat
- Anomalies
- Date and Time
- Procedure Step
- Tester's Name

The Incident Tracking report format can be changed according to industry standards and business requirements.

An example of one used in a company is:

Test Monitoring & Control in Software Testing

What is Test Monitoring?

Test Monitoring in test execution is a process in which the testing activities and testing efforts are evaluated in order to track current progress of testing activity, finding and tracking test metrics, estimating the future actions based on the test metrics and providing feedback to the concerned team as well as stakeholders about current testing process.

What is Test Control?

Test Control in test execution is a process of taking actions based on results of the test monitoring process. In the test control phase, test activities are prioritized, test schedule is revised, test environment is reorganized and other changes related to testing activities are made in order to improve the quality and efficiency of future testing process.
Congratulation! We now start with **Test Execution** phase. While your team works on the assigned tasks, you need to monitor and control their work activity.

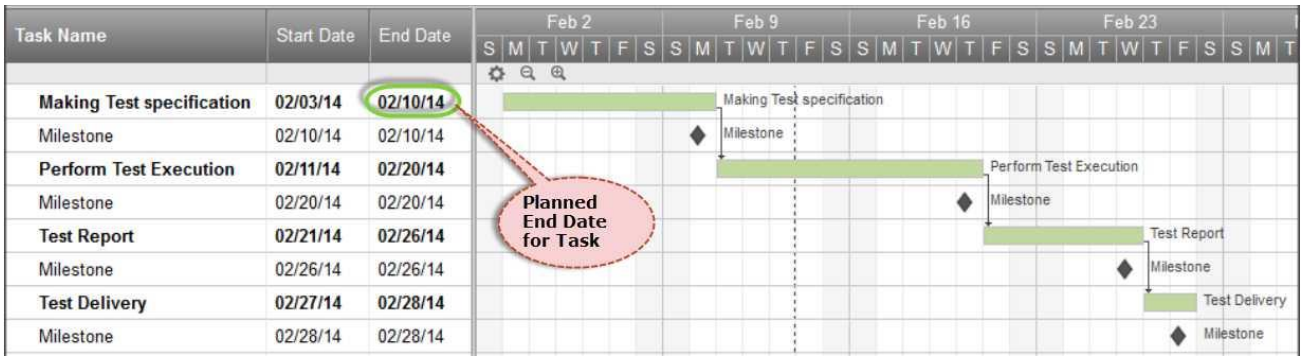
In the Test Management Phases tutorial, we briefly introduced Test Monitoring and Control. In this tutorial, you will learn it in detail.

Why do we monitor?

This small example shows you why we need to monitor and control test activity.

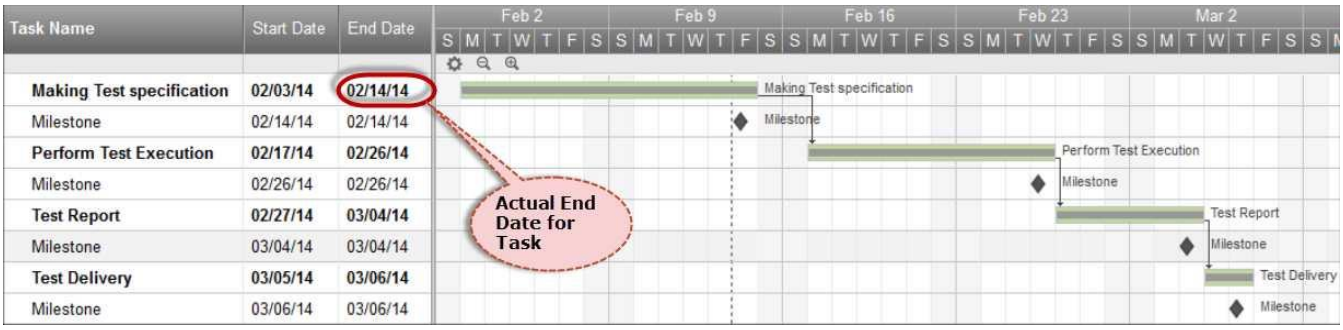
After finishing the Test Estimation and test planning, the management board agreed with your plan and the milestones are set as per the following figure.

Commands for Beginners Linux Tutorial15:03



You promised to finish and deliver all test artifacts of the Guru99 Bank Testing project as per above milestones. Everything seems to be great, and your team is hard at work.

But after 4 weeks, things are not going as per plan. The task of “Making Test specification” is **delayed by 4** working days. It has a cascading effect and all succeeding tasks are delayed.



You **missed** the milestone as well the overall project deadline.

As a consequence, your project fails and your company loses the customer trust. You must take full responsibility for the project’s failure.

Take a look the project progress, can you answer your boss's question

No matter how much and carefully we plan, something will go wrong. We need to actively monitor the project to

- **Early detect** and react appropriately to deviations and changes to plans
- Let's you communicate to stakeholders, sponsors, and team members **exactly** where the project stands and **determine** how closely your initial plan of action resembles reality
- It will be helpful for the Manager to know whether the project is going on the **right track** according to the project goals. Allows you to make the necessary adjustments regarding resources or your budget.

Project monitoring helps you avoid disasters. **Monitoring can be compared to checking the gas gauge in your car as you drive. It helps you see how much gas left in the tank, monitoring your project helps you avoid running out of gas before you reach your goal.**

What do we monitor?

Monitoring will allow you to make comparisons between your original plan and your progress so far. You will be able to implement changes, where necessary, to complete the project successfully.

In your project, as the Test Manager, you should monitor the key parameters as below

Cost

Costs are an important aspect of project monitoring and control. **You have to estimate and track basic cost information for your project**. Having accurate project estimates and a robust project budget is necessary to deliver project within the decided budget.

Suppose, your boss has agreed to fund the project with $100,000. You must keep an eye on the actual costs while the project is being implemented. As mentioned in Test estimation article, there's a ton of project activities which need money. You have to monitor and manage the project budget in order to control all that activities. Without monitoring the project cost, the project will most likely never be delivered on-budget.

Schedules

How can you work without a schedule? It can be compared to driving your car but without any idea of how long it takes you get to the destination. No matter how big or small is the size & scope of your project, you must prepare a project schedule. The schedule tells you

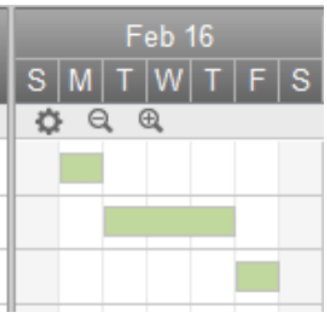
- When should each activity be done?
- What has already been completed?
- The sequence in which things need to be finished.

Here is an example of project schedule

You assigned a Team member to a Task: *Executing the Integration Cases of Guru99 Bank website*.

This task should be finished in one week. You can create a schedule as given below

Task Name	Duration	Start	Finish
Setup Test Environment	1	02/17/14	02/17/14
Perform Integration Test	3	02/18/14	02/20/14
Test Report	1	02/21/14	02/21/14



Resources

As mentioned in previous articles, **resources** are all things required to carry out the project tasks. They can be people or equipment required to complete the project activity. Lack of resources can affect the project progress.

The truth is, everything may not happen as planned, employees will leave, the project budget may be cut, or the schedule will get pushed. Monitoring resources will help you to early detect any resource crunch and find a solution to deal with it.

Quality

Quality monitoring involves monitoring the results of specific **work products** (like test case suite, test execution log), to evaluate whether its meets the defined quality standards. In case results do not meet quality standards, you need to identify potential resolution.

Example: Suppose that you monitored and controlled the project progress very well. Finally, you delivered the product at the deadline. The project seems to be successful.

But after delivering 2 weeks, you got this feedback from customer

Specialized testing for your enterprise application

Specialized testing for enterprise applications fills the gap that automated, standard testing cannot fill. The automated, standard testing does not have the necessary tools to conduct testing other than functional testing, load testing and distributed testing. Standard, automated testing does not require the skills of testers; it can be done by developers with minimal training in testing.

Specialized testing is a type of testing that you customize for an application -- from the perspectives of enterprise users. You can change criteria in the test in different scenarios for the same application. The source of criteria is a team of developers, users and business analysts. Examples include:

- Cloud testing to test scalability of enterprise applications
- User acceptance testing to test if user will accept the results
- Modular testing for certain modules of an application
- Agile testing to test against newly developed code with quality
- Incremental testing to test and add a module of the application

Example 1: Cloud testing

Cloud testing has been used in testing how well a cloud computing environment would be able to handle a spike in demand for an enterprise application service. This is made possible with Infrastructure as a Service (IaaS) that allows you to

extend the enterprise's data center on-demand. To get this type of cloud testing to work, the data center and the cloud computing environment must use the same virtualization platform. Since latency for WAN links are high, the transfer of the application to the cloud computing provider must be very quick.

Specialized cloud testing involves the use of test tools that are not provided by any vendors. The testers will need to develop tools that best meet the expectations of the developers, business analysts and users. For instance, you may want to use cloud testing to scale up to a specific audience, not all types of audience of users.

Example 2: User acceptance testing

User acceptance testing tests if user will accept the results. This type of testing starts with a plan and then a design and execution of test cases. Planning is the most important of all the steps. During the planning phase, you need to develop the user acceptance testing strategy to describe what criteria to use. Users should provide feedback to this strategy.

Test cases may be created from inputs from the use cases during requirements definition and a team of business analysts and the users. If defects in the application modules are found, you need to document the defects, how you resolve them and the results of fixing the problems. Make sure the users and the business analysts, as well as the specialized testers, are satisfied with the resulting outputs.

Example 3: Modular testing

Specialized modular testing focuses on the whether a module, the smallest testable portion, of an application works properly. You do not necessarily test each module individually or in sequence. You test the module in isolation for the desired outputs. You verify that each module's implemented structure matches the intended design structure. Once you find the module works, you can repeat the process of testing other modules.

Example 4: Agile testing

Agile testing emphasizes testing from the perspectives of users as early as possible. Testing is done as code becomes available and sufficiently stable and often as code becomes available and stable enough from module testing.

For instance, you can conduct test-driven development techniques when you use the extreme programming development method. This type of testing allows developers to write units tests before coding and ensures refactoring (design changes) does not break existing code. From the tests, the developers can write simplest design that will work and add complexity when needed. User acceptance tests are written before coding. You use reusable checklists to suggest tests and listen to user stories of features that need to be added. Tests should be in a format that is easy to understand by customers or users.

Example 5: Incremental testing

Incremental testing is more than modular testing. With incremental testing, you test each module of the software individually and then add another module to continue testing. This is useful when you are trying to find an error in a haystack that you would not be able to find if you test the entire application. You could change test data or criteria for a particular module to find out what the results would be.

Let's suppose you have 1000 modules of an application. You want to test how module no. 20 works with different test data and show the results to users. When you first start, you test the first module and then add the second module to continue the test. When you get to module no. 20 and users are not satisfied with the results, you can back track to a module of issue, make changes to the module, and repeat the process of incremental testing.