Testing Strategy

Testing will be done in continuous iteration and all the data will uploaded to the GitHub repository. Testing will happen in four phases:

- Test Planning
- Test Analysis
- Test Design
- Test Execution
- Test Closure

Test Planning

This step will comprise of the planning for the future testing, Test policy, plan and test strategy. On the basis of the project, and code and class implementation panning process will occur. All the tools and features that will used in the further testing will be decided in this part. The amount of time to be spent and member to be allocated will also be decided ahead of the testing process. This step will come as a pillar for the upcoming testing steps.

The following points will be kept in mind:

- Always implement highest priority work items first (Each new work item is prioritized by Product Owner and added to the stack).
- Work items may be reprioritized at any time or work items may be removed at any time.
- A module in greater detail should have higher priority than a module in lesser detail.

Planning for the testing will amount to 3-5 hours of work.

Identify what is included in testing for this particular project. Consider what is new and what has been changed or corrected for this product release.

- (Automated) Unit testing
- Code analysis (static and dynamic)
- Integration testing
- (Automated) Feature and functional testing
- Data conversion testing
- System testing
- (Automated) Security testing
- Environment testing
- (Automated) Performance and Availability testing
- (Automated) Regression testing
- Acceptance testing

Test Analysis

This step of the process will there to find out the requirements and clarity regarding the analysis of the project. For the testing of the whole project and sources codes, various testing tools are used like Junit, which is a unit testing framework for the Java programming language.

Different Tools and Procedures that will be used are:

Testing type	Description	Tool to be Used
Unit testing	Testing that verifies the implementation of software elements in isolation	JUnit
System Testing	Testing the whole system with end to end flow	Selenium, QTP
Functional and Feature Testing	Testing an integrated hardware and software system to verify that the system meets required functionality	Selenium, QTP
Code analysis (static and dynamic)	Walkthrough and code analysis	AgileJ and DynInst
Integration Testing	The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.	Vector Cast

All the codes and the tool information will posted, once the first iteration of testing is completed.

Test Design

This step is the next step where the analysis provided in the last step will be transferred on as a design. Testing all the prior features and re-testing previously closed bugs will be done by creating a test design. Testing based on acceptance criteria to enable the customer to determine whether or not to accept the system will included in this section.

The following are the strategies:

- Specification based / Black box techniques (Equivalence classes, Boundary value analysis, Decision tables, State Transitions and Use case testing)
- Structure based / white box techniques
- Experience based techniques (Error guessing and Exploratory testing)

Test Execution

This step will comprise of the actual execution of the testing process. All the tool and features discussed will be used to create a test for the whole project. Data management will require additional testing strategy.

We will keep in mind the following points:

- Agile testing must be iterative.
- Testers cannot rely on having complete specification.
- Testers should be flexible.
- They need to be independent and independently empowered in order to effective
- Be generalizing specialists.
- Focus on What and Not How to test.
- Shorter feedback cycles
- Focus on sufficient and straightforward situations.
- Focus on exploratory testing

Key execution steps could include:

- Steps to build the system
- Steps to execute automated tests
- Steps to populate environment with reference data
- Steps to generate test report/code metrics

The user interface, features and data management will go through separate testing processes. By using the above-mentioned tools, the whole process of testing execution can be done within 1 week.

The automatic testing will be implemented keeping the following things in mind:

- Risk
- How long it takes to run the tests manually?
- What is the cost of automating the test?

- How easy are the test cases to automate?
- How many times is the test expected to run in project?

Test Closure

Test closure will include collecting the testing result data and codes that were implement, for the future iterations for the testing process. Finally calculating the result and acquiring whether the project will need any improvement, additional code or another test iteration.

This will give a summary of all the tests conducted during the software development life cycle, it will also give a detailed analysis of the bugs removed and errors found. In other words, Test Closure is a memo that is prepared prior to formally completing the testing process.

Test closure activities fall into four main groups:

- Test completion check - ensuring that all test work is indeed concluded. For example, all planned tests should be either run or deliberately skipped, and all known defects should be either fixed and confirmation tested, deferred for a future release, or accepted as permanent restrictions.

- Test artifacts handover - delivering valuable work products to those who need them. For example, known defects deferred or accepted should be communicated to those who will use and support the use of the system. Tests and test environments should be given to those responsible for maintenance testing. Regression test sets (either automated or manual) should be documented and delivered to the maintenance team.

- Lessons learned - performing or participating in retrospective meetings where important lessons (both from within the test project and across the whole software development lifecycle) can be documented. In these

meetings, plans are established to ensure that good practices can be repeated and poor practices are either not repeated or, where issues cannot be resolved, they are accommodated within project plans.

- Archiving results, logs, reports, and other documents and work products in the configuration management system. For example, the test plan and project plan should both be stored in a planning archive, with a clear linkage to the system and version they were used on.