Test strategy FOR [ShipRealWind Calculator]

Project name: ShipRealWind Calculator 3.0

Document control

|  |
| --- |
| Version: 1.0 |
| Date: 05.05.2023 |
| Developed by: ELENA-GRAȚIELA ROBE-VOINEA |

Document sign-off

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Status | Date | Approved by | Job Title |
| 1.0 | Final |  | [ADNANA CORDOȘ] | PRODUCT OWNER |

## Contents

## [1](#_heading=h.gjdgxs) introduction

## [2](#_heading=h.30j0zll) purpose

## [3](#_heading=h.1fob9te) system overview

## [4](#_heading=h.3znysh7) scope of testing

## [4.1](#_heading=h.2et92p0) In scope

## [4.2](#_heading=h.tyjcwt) out of scope

## [5](#_heading=h.3dy6vkm) principles & application

## [5.1](#_heading=h.1t3h5sf) project team

## [5.2](#_heading=h.1y810tw) [test phase 1] - first sprint

## [5.3](#_heading=h.qsh70q) [Test Phase 2] - second sprint

## [6](#_heading=h.206ipza) environment requirements/Production Environment

## [7](#_heading=h.2dlolyb) test data requirements

## [8](#_heading=h.1rvwp1q) testing tools & techniques

## [9](#_heading=h.25b2l0r) team roles & responsibilities

## [10](#_heading=h.2iq8gzs) defect management

## [11](#_heading=h.1x0gk37) test schedule

## [12 referenced documents](#_heading=h.1302m92)

## 

# introduction

Information technology is the word of the moment and absolutely everything we conceive, analyze and implement is directly related to this concept. Maritime transport is an important pillar for our days, a decision factor for the global maritime market, and the connection between the two is measured in time and money. For a better efficiency in calculating real wind and its components, a computational method using dedicated software programs is presented.

Our application was designed with the help of Python and Qt Designer, and it proved to be more challenging than we initially anticipated. For this particular app, we used a Vaisala weather station, just like the ones on the ship.

The objective of the ShipRealWindCalculator project was to create a graphical user interface for solving the problem of calculating the components of real wind (real wind direction and real wind speed) from the composition of ship wind and apparent wind. Although at first glance it may seem like a simple problem, putting it into practice involves several challenges given that the application is at the intersection of the sailing navigation and mathematical domains.

# purpose

This Test Strategy will provide a high-level view of how testing will be completed for the ShipRealWind Calculator project. There are many sections to this document, but the focus is on the overall approach to testing for the different test phases that are to be completed.

# system overview

The System Under Test (SUT) consists of the following components:

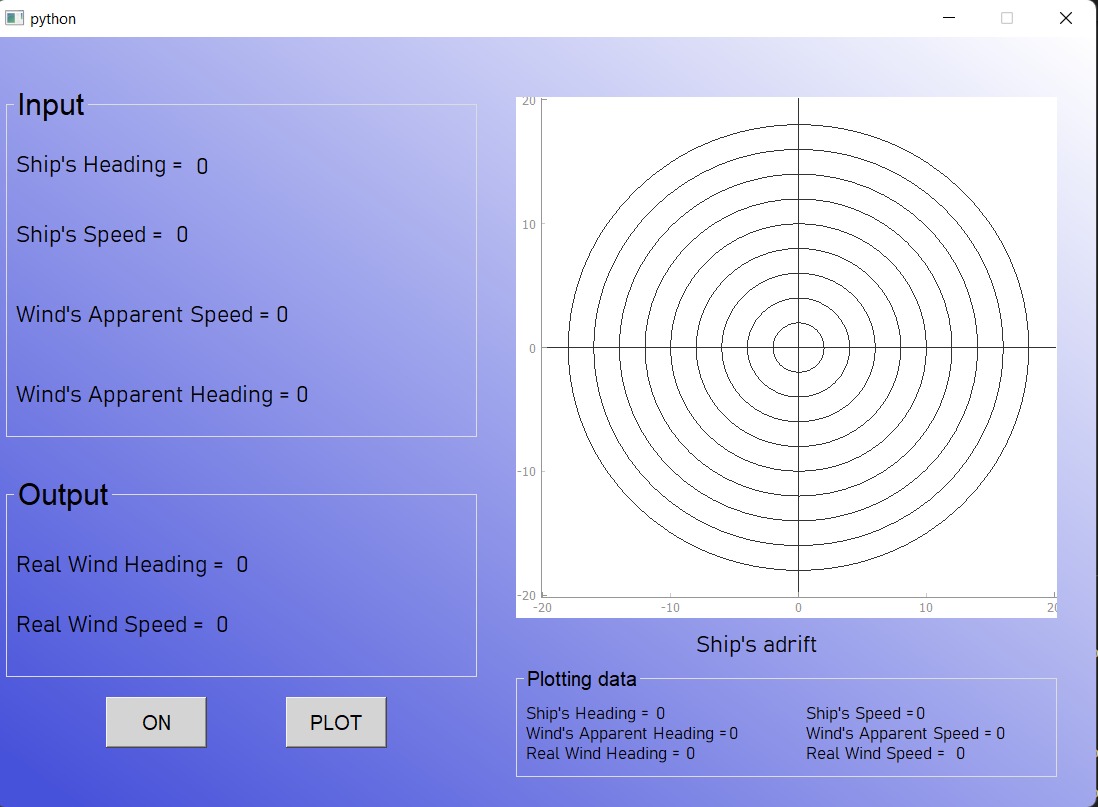


Fig. 1 – Main Window ShipRealWindCalculator

Fig. 2 - Architectural Diagram of the SUT

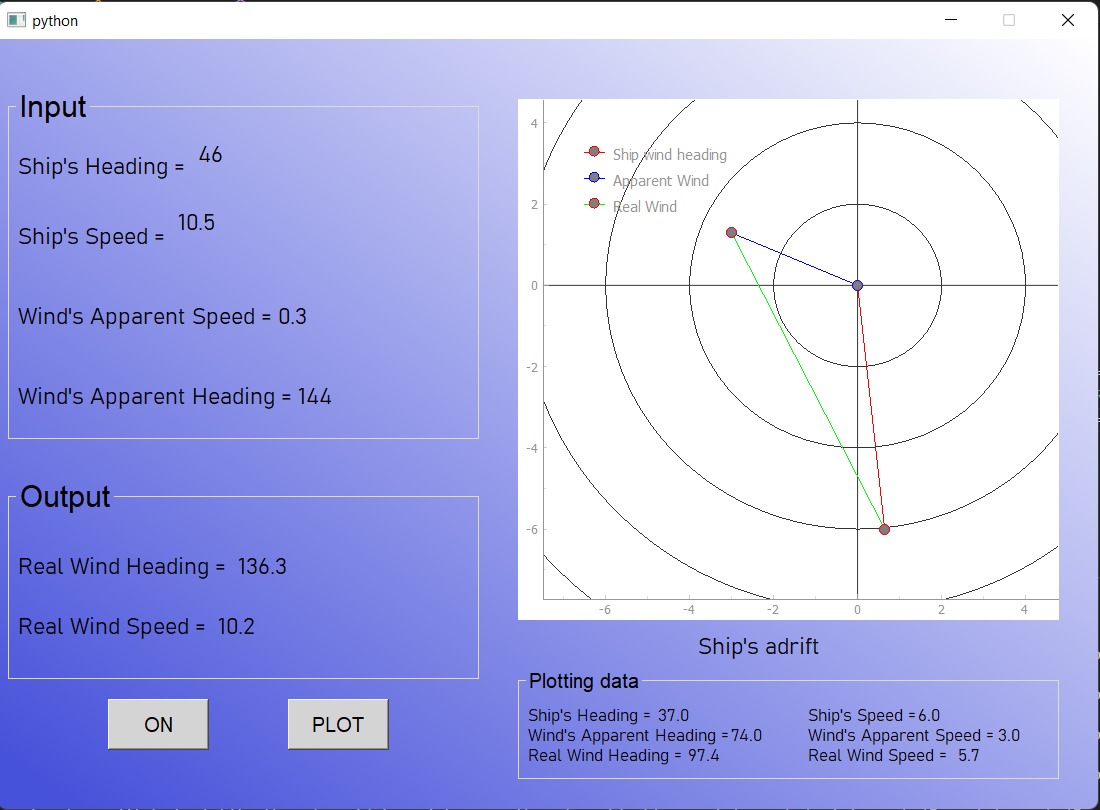


Fig 3. – Main window ShipRealWind Calculator afert execution

# scope of testing

As the diagram (Fig.2) in section 3 shows, the scope of the ShipRealWind Calculator testing is to check as much as possible of the components in order to assure that the application is properly functional and ready to use. Therefore functionalities in scope are grouped together in a set of work-stream processes approved for our applications. The below mentioned are the work-stream processes in scope along with the applications listed. All these work-stream processes will be part of the two sprints that will be performed.

|  |  |  |  |
| --- | --- | --- | --- |
| **Work Stream to test** | **Operating system** | | **Environment (where to test)** |
| Input Parameters | Windows | Raspberry Pi 3 | Local -ShipRealWindCalculator 3.0 |
| Output Parameters | Windows | Raspberry Pi 3 | Local -ShipRealWindCalculator 3.0 |
| Data Plotting Parameters | Windows | Raspberry Pi 3 | Local -ShipRealWindCalculator 3.0 |
| “On” button functionability | Windows | Raspberry Pi 3 | Local -ShipRealWindCalculator 3.0 |
| “Plot button” button functionability | Windows | Raspberry Pi 3 | Local -ShipRealWindCalculator 3.0 |
| Plot design | Windows | Raspberry Pi 3 | Local -ShipRealWindCalculator 3.0 |

## In scope

The types of testing (level of testing) that are in-scope for our app are as described below:

Fig. 4 – type and level of testing applicable for ShipRealWind Calculator 3.0

## out of scope

The types of testing that are out of scope are:

* Component/Unit Testing and Component/Unit Integration Testing is not in scope for Testing.
* Any third-party application testing that is not mentioned in the application scope.
* Security testing is out of scope
* Any code changes or configuration fixes are out of scope for testing.
* Defect fixes are out of scope from the Testing, except for the defects that are relevant to the test case changes.

## principles & application

## 

It is important to understand the risks to the business with regards to any system that is developed. Some parts of the system will be critical to the running of the business and some will not. By understanding the risk profile, we can tune the amount & types of testing we complete. This can then give a realistic scale of testing for each project and therefore we can support the estimates we put forward.

Ship Realwind Calculator adheres to the “7 principles”of testing, as follows:

**Principle 1 – Testing shows presence of defects**

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the developed solution but, even if no defects are found, it is not a proof of correctness.

**Principle 2 – Exhaustive testing is impossible**

Testing of all combinations of inputs and preconditions is not feasible, except for trivial cases. Instead of exhaustive testing a risk analysis and priorities should be used to focus testing efforts.

**Principle 3 – Early testing**

To find defects early, testing activities shall be started as early as possible in the software development lifecycle and shall be focussed on defined objectives.

**Principle 4 – Defect clustering**

Testing effort shall be focused proportionally on the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.

**Principle 5 – Pesticide paradox**

If the same tests are repeated over and over again, the same set of test cases will no longer find any defects. To overcome this “pesticide paradox”, test cases need to be regularly reviewed and revised and new and different tests need to be written to exercise different parts of the software or system to be able to detect potentially more defects.

**Principle 6 – Testing is context dependent**

Testing is done differently in different context. For example, ShipRealWind Calculator it is tested differently depending on which workstation and operating system has the tester that make the test.

**Principle 7 – Absence-of-errors fallacy**

Finding and fixing defects does not bring value if the system built is unusable and does not fulfil the users’ needs and expectations.

## project team

TEAM defines six dimensions, the core dimensions of: Process, Team and Tools and the supporting dimensions of: Communication, Culture and People. Each of these is addressed specifically in the testing discipline within the methodology.

There are two distinct software development approaches described by TEAM:

* Agile
* Planned Iterative

This project will be delivered using the Planned Iterative methodology.

Planned Iterative Principles impacting testing include:

* Testing is completed by an Independent Testing Group
* Test throughout the Project
* Focus on finding defects early in the SDLC.
* Formal Testing Scope within each iteration (Continuously verifying quality)
* Focus on testing the Architecture first.
* Test Cases derived from Use Cases.
* Tailor process and documentation for the size and complexity of the project

## [test phase 1] - first sprint

### Objective

The objective of [Test Phase 1] is to verify that the application works in terms of input, output and plotting parameters both on the Windows operating system and on the Raspberry Pi 3 system in optimal conditions.

### Scope

The scope of testing for [Test Phase 1] is to test the Input, Output and Plotting parameters of the application ShipRealWind Calculator 3.0 in accordance with the type of testing mentioned in Fig. 4 from subsection 4.1 as follows:

* Functional Testing (System testing>>Black Box testing)

This type of functional testing evaluates the complete software application as a whole. With system testing, the goal is to ensure that the software application meets the functional requirement. System testing is performed manually . include both testing the user interface as a whole or testing the performance of the application itself.

In much more detail, we can refer here to a Functional Black Box testing type because this requires no working knowledge of an application’s internal functions or processes. Testers execute predetermined test cases to identify functionality issues rather than internal implementation problems. This is purely focused on inputs and outputs, as our user story ask.

* Acceptance Testing (Alpha Testing)

Acceptance Testing also lends itself very well to this Sprint 1 because this type of functional testing is used to evaluate whether an application or product meets the stakeholders' or customers' acceptance criteria and requirements. It involves testing the application on different use cases and verifying it meets the functional and non-functional requirements along with user expectations and needs. In our case the user can be consider the two OS namely the Windows and Raspberry Pi 3 .

Due to the fact that I am also in the position of application developer, Alpha Testing is by default to be followed because as it is defined, this is one of the software testing types in software engineering conducted in-house by developers or testers to evaluate an application before it is released to the public.

* Non-Functional Testing (Usability, Exploratory, Compatibility Testing)

Non-functional aspects of a software application such as: usability, exploratory and compatibility testing they are also defining what concerns the test cases in this sprint.

The actual interaction of the user with the application is naturally considered **usability testing**, because this type of non-functional testing evaluates how user-friendly and easy-to-use a software application is. This can involve observing your users as they interact with an application, allowing identification of issues or challenges users may encounter, such as slow response times, difficulty completing tasks, or confusing user interfaces.

On a different note, if we were to report what **Exploratory testing** means applied to our application it means that this involves testing software without predefined test cases or scripts by exploring and experimenting with an application. This helps identify errors or bugs that were not detected by existing predefined tests while gaining a better understanding of the functionality and performance of an application. We can say that we consciously do this during the entire sprint

Somewhat similar to Acceptance Testing, the testing concepts of Compatibility Testing

can be considered also satisfied by the test cases of this sprint because This type of non-functional software testing checks whether an application can function as expected with different hardware, operating systems, browsers, databases, and other software components. This helps to make the application function seamlessly on different platforms to ensure a consistent user experience.

* Others (Installation, Configuration, User Interface Testing)

Aspect that involves testing the visual design, layout, and responsiveness to ensure a consistent and positive user experience are part of the User Interface testing and together with Installation and Configurations should be considered others types of testing or rather, all of the above but viewed from a different perspective, strictly focused on the operability of the application from the strict point of view of the Input, Output and plotting parameters

### Test Preparation

The testing process follows from the strategy approach: it defines phases of testing, relationships between them and their entry and exit criteria.

#### Entry Criteria/DOR

* The business value is clearly articulated (in the format of “As [a type of user] I want [some] goal so that [some reason]”).
* The story follows the INVEST model.
* The story has a 2 – 3 word short summary.
* The story is small enough to fit in one sprint.
* The story has clear and concise acceptance criteria which describe all of the features of the story. Details are captured as a narrative texts that describe an interaction of the user and the system, focusing on the value a user gains from the system.
* Once the acceptance criteria have been met the story is complete.
* No external dependencies block the story being completed.
* Story identifies external expertise and provides contact details

#### Exit Criteria/DOD

* Tests written and passing
* Documentation updated
* Acceptance criteria met
* Tested on the following devices:
  + Windows
  + Raspberry Pi 3 model B + MicroSD Card Noobs Raspberry Pi V3 16 GB

## [Test Phase 2] - second sprint

### Objective

The objective of [Test Phase 2] is to verify that the application works both from a functional and non-functional point of view as we describe in the next section.

### Scope

The scope of testing for [Test Phase 2] is to test the "ON" button, the "PLOT" button on Windows and Raspberry PI OS’s as well as the area dedicated to the plot graph just on Windows OS.

The type of testing for the application ShipRealWind Calculator 3.0 are in accordance with the ones testing mentioned in Fig. from subsection 4.1 as follows:

* Functional Testing (System testing>>Black Box testing)

This type of functional testing evaluates the complete software application as a whole. With system testing, the goal is to ensure that the software application meets the functional requirement. System testing is performed manually . include both testing the user interface as a whole or testing the performance of the application itself.

In much more detail, we can refer here to a Functional Black Box testing type because this requires no working knowledge of an application’s internal functions or processes. Testers execute predetermined test cases to identify functionality issues rather than internal implementation problems. This is purely focused on inputs and outputs, as our user story ask.

* Acceptance Testing (Alpha Testing)

Acceptance Testing also lends itself very well to this Sprint 2 because this type of functional testing is used to evaluate whether an application or product meets the stakeholders' or customers' acceptance criteria and requirements. It involves testing the application on different use cases and verifying it meets the functional and non-functional requirements along with user expectations and needs. In our case the user can be consider the two OS namely the Windows and Raspberry Pi 3 .

Due to the fact that I am also in the position of application developer, Alpha Testing is by default to be followed because as it is defined, this is one of the software testing types in software engineering conducted in-house by developers or testers to evaluate an application before it is released to the public.

* Non-Functional Testing (Usability, Exploratory, Compatibility Testing)

Non-functional aspects of a software application such as: usability, exploratory and compatibility testing they are also defining what concerns the test cases in this sprint.

The actual interaction of the user with the application is naturally considered **usability testing**, because this type of non-functional testing evaluates how user-friendly and easy-to-use a software application is. This can involve observing your users as they interact with an application, allowing identification of issues or challenges users may encounter, such as slow response times, difficulty completing tasks, or confusing user interfaces.

On a different note, if we were to report what **Exploratory testing** means applied to our application it means that this involves testing software without predefined test cases or scripts by exploring and experimenting with an application. This helps identify errors or bugs that were not detected by existing predefined tests while gaining a better understanding of the functionality and performance of an application. We can say that we consciously do this during the entire sprint

Somewhat similar to Acceptance Testing, the testing concepts of Compatibility Testing

can be considered also satisfied by the test cases of this sprint because This type of non-functional software testing checks whether an application can function as expected with different hardware, operating systems, browsers, databases, and other software components. This helps to make the application function seamlessly on different platforms to ensure a consistent user experience.

* Others (Installation, Configuration, User Interface Testing)

Aspect that involves testing the visual design, layout, and responsiveness to ensure a consistent and positive user experience are part of the User Interface testing and together with Installation and Configurations should be considered others types of testing or rather, all of the above but viewed from a different perspective, strictly focused on the operability of the application from the strict point of view of the Input, Output and plotting parameters

### Test Preparation

The testing process follows from the strategy approach: it defines phases of testing, relationships between them and their entry and exit criteria.

#### Entry Criteria/DOR

Entrance criteria are the required conditions and standards for work product quality that must be present or met prior to the start of a test phase.

**According with Agile approach(Scrum)**

**Test Planning:**

• User Story with clear and univocal Acceptance Criteria is available before test planning.

• Defined and shared Definition of Done.

**Test Execution:**

• No open critical/major defects impacting functionality (User Story) under tests.

• Development of all items to be tested is completed and deployed on test environment.

• Testing environment is configured and ready.

* + Windows
  + Raspberry Pi 3 model B + MicroSD Card Noobs Raspberry Pi V3 16 GB

#### Exit Criteria/DOD

The set of generic and specific conditions, agreed upon with the stakeholders for per-mitting a process to be officially completed. The purpose of exit criteria is to prevent a task from being considered completed when there are still outstanding parts of the task which have not been finished. Exit criteria are used to report against and to plan when to stop testing.

More detailed criteria are identified by project manager after consultation with busi-ness team, analyst and test team on a specific project.

According with Agile approach(Scrum) the DOD respect the following:

* Exit criteria define when the User Story can be considered as closed, and ready to be added to the current increment. They shall include:
* Successful execution of the Test Script (common approach is one Test Script per User Story).
* Meeting all Acceptance Criteria (agile approach allows changes in AC when agreed with Product Owner).
* No open critical, major or average severity defects (unless the issue is de-termined to be low impact and low risk).

According to Scrum, above exit criteria should be part of definition of done and should be confirmed by entire Team.

# environment requirements/Production Environment

The Test environment(s) will be configured as shown below:

[Overview of System Infrastructure Components]

**UI** – The user interface is among the most important software application parts. In our case is the desktop application ShipRealWind Calculator 3.0.

Just as a UI is always used at every part in the backend, this remains valid also in the front end when it comes to testing the application, therefore one of the core elements of software testing is the data you use to test a particular software.

**Manual tester** - a manual software tester who check for the new application quality without automation testing tools or scripting processes.

**Operating systems**-an operating system (OS) is nothing but a program that you load into the computer through a boot program. It would manage every application program on your computer.

For this desktop app as we mentioned before it will be used for testing Windows and Raspberry Pi OS.

[Who manages the Environments?]

In general terms for the design and implementation of the test environment there is a dedicated person called Test Environment Manager. This person is responsible for designing and implementing the test environment, including hardware, software, network, and data configurations. In our case the environments is managed by the tester himself.

[Refer to Test Environments documentation]

<https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf>

# test data requirements

[What data is used for testing purposes?]

|  |
| --- |
| **Test input Data** |
| Input parameters |
| Output parameters |
| Plotting parameters |
| On button |
| Plot button |

[Test Input Data & Test Reference Data?]

|  |  |  |
| --- | --- | --- |
| **No.** | **Test input Data** | **Provided by** |
| 1 | Input parameters | 2 from the back end , 2 from in real time from the weather station |
| 2 | Output parameters | All 4 from the back end |
| 3 | Plotting parameters | Backend |
| 4 | On button | User interface+backend |
| 5 | Plot button | User interface+backend |
| 6 | Plot area | User interfae+backend |

[How do the different phases use different Datasets?]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Test input Data** | **STLC** | | | | | |
|  |  | **Requirement**  **Analysis** | **Test**  **Planning** | **Test**  **Case development** | **Environment**  **Setup** | **Test**  **Execution** | **Test cycle closure** |
| 1 | Input parameters | x | x | x | x | x | X |
| 2 | Output parameters | x | x | x |  | x | X |
| 3 | Plotting parameters | x | x | x | x | x | X |
| 4 | On button | x | x | x | x | x | X |
| 5 | Plot button | x | x | x | x | x | X |
| 6 | Plot area | x | x | x | x | x | X |

# testing tools & techniques

The tools inside the testing discipline are in a permanent process of evaluation and customization so that they can offer the best solution in the right context; however the tools described in the following sections are proposed for use in this project.

Testing tools:

* Jira
* TestCaseLab
* PowerToys
* Bandicam
* Windows Screen Ruler
* Kazam screen recorder for Raspberry Pi
* Testing techniques: the ones mentioned in section 4.1, fig. 4.

## Requirements & Use Case Management

The requirements and use case management are estsblished in the User Stories, and provided in the acceptance criteria.For writing the US it was used the Jira tool management and for the use cases management and writing TestCaseLab tool. The TR , matrix and reports were developed also in the TesCaseLab tool. For all mentioned above, US, TC and TR the we considered the following

* Provide test case properties, including ID, Name, description.
* Display preconditions, the expected result, and test data (if defined).
* Deliver concise test case instructions (if defined).
* Present test case results, if available.
* Show all related requirements and risks.

## Test Management & Defect Tracking

If a test is considered to have failed, a corresponding defect must be logged against it using defect tracking tool. This should be done by a person who executed this test or encountered the failure in any other way (e.g. Using the application). Defect statis-tics can be used to make testing scope decisions. For this project it was used Jira tools for test management and defect tracking.

# team roles & responsibilities

The following table shows the testing roles for the project, together with the individuals involved in the testing effort.

|  |  |  |
| --- | --- | --- |
| Activity | Responsibility/Ownership | Name |
| Test Plan Creation | [Test Manager] | E.G.Robe |
| Test Phase Plan Creation | [Test Lead] | E.G.Robe |
| Test Management | [Test Lead] | E.G.Robe |
| [Test Analysis and Design] | [Test Lead] | E.G.Robe |
| [Test Preparation, Execution & Results] | [Test Lead] | E.G.Robe |
| [Test Defect Submission] | [Test Lead] | E.G.Robe |
| [Test Summary Reporting] | [Test Lead] | E.G.Robe |
| [Test Completion Reporting] | [Test Lead] | E.G.Robe |

# defect management

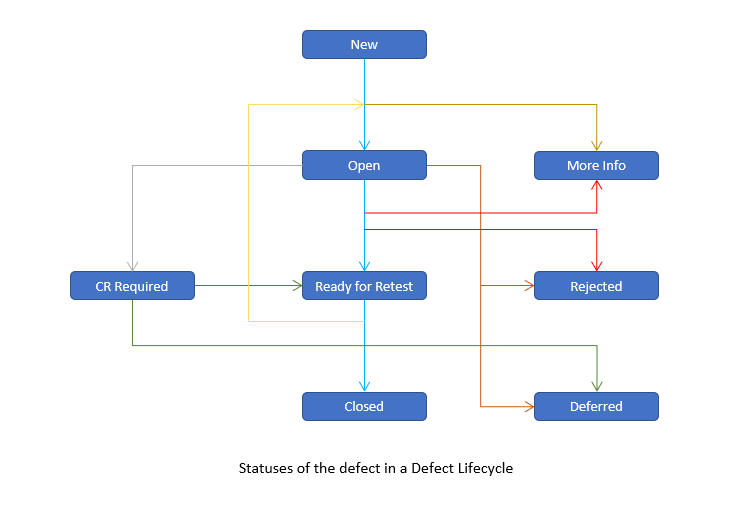
A defect life cycle process explains the various statuses of the defects throughout its life time.

Any deviation from the specification mentioned in the functional requirements calls for a defect. A defect is raised when the Expected Result of the test does not meet the Actual Result. As a result, the product does not meet the end user expectation or the functional requirements.

While there are various reasons on why the defects occur in the system, below are the statuses that the defect would undergo once raised in the test management tool during the defect life cycle management.

|  |  |
| --- | --- |
| Defect Status | Description |
| New | Defect is identified and just raised by the tester |
| More Info | Defect requires additional info for analysis and acceptance by dev or functional team |
| Open | Defect is accepted and assigned to a dev or functional team for a fix |
| Ready for Retest | Configuration and code changes related to the defect are moved to the Quality system and the defect is ready to be tested |
| Rejected | Defect is Rejected by the work-stream |
| Closed | Defect is fixed and tested successfully, and no further action required |
| CR Required | A defect requires an additional development or new configuration to be implemented. Set by the work-stream |
| Deferred | This status of the defect is dependent on the *CR required* status. |

Defect lifecycle is explained in the below flow chart



## defect management process/status

* There will be few cases where the defects raised for a functionality might call for a new requirement to be created. In that case, the requirement must go through the whole process of analysis, creation, approval, design, build and test.
* The test on which this new requirement dependent defect is created, can be put to ‘Passed’ status as the test meets the original requirements based on which the test was created.
* Check with the stakeholders for the possibility of implementing the new requirement (based on the defect) within the same test phase and include the new Requirement ID details in the defect, before closing the defect.
* If there a defect identified on the Quality/Pre-Prod system which also applies to the production system, then an incident to be raised on the production system and the defect must be updated with the incident details. The relevant test that is blocked by this defect must be in failed status and a clear statement should go in the Test Closure Report.

# test schedule

### 

The following table shows the high level testing milestones for this phase of the project.

|  |  |  |  |
| --- | --- | --- | --- |
| Ref No | Stage | Project Milestone | Due date |
| TM101 | Test Preparation Stage | Phase Test Plan document completed. | 01/05/2023 |
| TM102 | Test Preparation Stage | Test analysis completed on the detailed requirements and technical documentation. | 01/05/2023 |
| TM103 | Test Preparation Stage | Test Conditions/Cases/Scripts completed and signed off. | 01/05/2023 |
| TM104 | Test Execution Stage | Execution of Test scripts completed. | 01/05/2023 |
| TM105 | Deliverable | Matrix | 04/05/2023 |
| TM106 | Deliverable | Arhitecture | 04/05/2023 |

# referenced documents

The following table identifies the documentation used for developing this Test Plan:

|  |  |  |  |
| --- | --- | --- | --- |
| # | Document | Author | Description |
| 1 | Test Plan | E.G Robe | This document provides information regarding what specific testing will be completed on the Project. |
| 2 | TCR 1 | E.G Robe | This provides information with regards to the Sprint 1 |
| 3 | TCR 2 | E.G Robe | This provides information with regards to the Sprint 2 |
| 4 | Matrix | E.G Robe | Matrix reports, Conclusions |
| 5 | Arhitecture | E.G Robe | App architecture |