RESOURCE MANAGER

MASTER TEST PLAN

ITC303 – BAMJ SYSTEMS

Contents

[1. Management summary 1](#_Toc12295733)

[2. Test objectives 1](#_Toc12295734)

[3. Test Approach 1](#_Toc12295735)

[4. Results and Deliverables 1](#_Toc12295736)

[5. Test Environment 2](#_Toc12295737)

[6. Test Process Management 2](#_Toc12295738)

[7. Defects Procedure 3](#_Toc12295739)

[7.1 Defect Identification 3](#_Toc12295740)

[7.2 Defect Resolution 3](#_Toc12295741)

[8. Test Strategy 4](#_Toc12295742)

[8.1 Test Objectives 4](#_Toc12295743)

[8.2 Product Risk Analyses 4](#_Toc12295744)

[8.3 Strategy 5](#_Toc12295745)

1. Management Summary

The Testing component of Resource Manager is one of the most important aspects to ensuring that requirements and usability needs are met with the expectations of the Project Sponsor.

This document, at a high level, will outline how to be successful and complete the testing phase for Resource Manager.

1. Test Objectives

The objectives of testing the Resource Manager system are to ensure that the most common use cases can be completed in multiple ways, without resulting in any errors.

If errors do occur, it is meaningful that the person performing the action should understand what caused the problem in order to allow the desired outcome to be successfully obtained.

Current key Use Cases:

* Users updating their skill profile
* Project managers assigning resources
* Project managers create projects

1. Test Approach

The performance of testing will begin towards the end of the Construction phase. This will allow any necessary last-minute adjustments before shipping the final release.

There will be a variety of testing to each component such as unit testing, functional testing and user acceptance testing. This will be followed by completing a final functional test before the product is deemed complete.

All tests performed by the project team will be conducted in order to meet specific testing criteria. This will ensure the required functionality has been tested before handing functional testing over to user acceptance.

1. Results and Deliverables

Test cases are expected to have a signed Test Acceptance document from both the internal tester as well as the key stakeholder for user acceptance testing. This must include log from the application (unit testing) or screenshots of the tests that have been performed for internal functional testing.

User acceptance testing will require an acceptance certificate.

1. Test Environment

The project team will conduct a variety tests that involve many devices and technologies. This will best demonstrate the Resource Manager project’s functionality and ability to meet the useability requirements.

Unit and integration testing will be done using JUnit and Mockito Testing within Eclipse as this is the platform the project will be developed on and all team members are familiar using this technology.

EclEmma is an Eclipse plugin for code coverage so we can make sure enough code paths are being tested.

User Acceptance testing will be done by the project team and the client. All team members of the project will be using their own personal devices and testing the system using Google Chrome and Firefox. The Client will use their own system for testing.

The full testing infrastructure in use is summarised below.

|  |  |  |
| --- | --- | --- |
| **Tools** | **Description** | **Version** |
| Eclipse/JUnit/Mockito/EclEmma | Eclipse is used for writing/reading code, Junit/Mockito to write test programs for Unit/Integration testing. EclEmma for code coverage. | Eclipse Java IDE 2019-03 (may be updated as project proceeds)  Junit 4 |
| Windows OS | All testing devices will be running on Windows OS. | Windows 8/10 |
| Google Chrome/Mozilla Firefox | Primary web browsers used for testing. May also include Internet Explorer/Microsoft Edge. | Latest |
| Trello | Software used to monitor and log test results and outstanding issues. | Latest |

1. Test Process Management

The management of test processing can be divided into three parts:

1. **Test Analysis**: An analysis of the functional and non-functional requirements is carried out by the test team. The creation of tests will be completed prior to prioritising them by their respective importance. These test cases ensure that the system, integration and UAT has been acknowledged as a pass.
2. **Test Planning**: The test cases, test data and testing environment preparations are carried out. To outline the scope and objectives to be completed, a test plan is prepared, including scenario test cases within the system. The test cases will outline the steps required to test certain functionality and/or areas of the system and will include the expected and actual results. Preparation of the test environment is an essential part of this phase as the testers will not be able to test the system without it. Once coding is complete, a build will be prepared and pushed to the tester’s environment where the execution of test cases will be performed.
3. **Test Execution**: The testers will execute the test cases that were prepared in the ‘Test Planning’ phase. During the execution, if any expected and actual results do not match then the tester will open the bug(s) as a new defect within Trello, bring them to the attention of the developers to fix. Bug reporting/fixing should follow a complete ‘Defect Procedure’ life cycle.

Once the test cases have been carried out and there are no pending critical bugs to be fixed, all test cases and bug reports (if any) must be reviewed and marked as approved/denied. If accepted, the system will be logged as ready for release.

The developer team will then proceed with analysing the things that went well and the potential bugs that may have been found. From this data, they can decide which areas need improvement or not and get prepared to push the release of the system.

1. Defects Procedure

A defect is a bug or fault in the system that has been brought to the attention of the development team and has been accepted as a defect after team analysis.

7.1 Defect Identification

By identifying defects early in the project, the cost of the overall system will be decreased as bugs found later (especially after release) can cost more to fix/prevent.

There are three steps involving defect identification:

1. **Finding the defect**: Identify the bug/defect before it becomes a major problem in the system.
2. **Reporting/logging the defect**: When the testing team identifies an issue, they must make the development team aware that there is an issue that needs to be acknowledged and corrected. The testers will use Trello to log and track the status of issues. Developers can also use this platform to see possibly defects that have been identified by the users.
3. **Confirming the defect**: Once the testers have nominated a potential bug/defect, the development team will acknowledge and confirm the status of the defect. If it is accepted as a defect, it will be fixed and prevented from happening again, otherwise it will be marked as not a defect.

7.2 Defect Resolution

As the testing team has identified the bugs/defects, the development team now needs act on resolving the issue(s).

There are three steps to resolving defects:

1. **Prioritising the risk**: Defects will be prioritised by the development team according to their potential impact on the system. Issues that could have a high impact will be high priority and issues that may not have an effect to general functionality will be placed as low priority.
2. **Fixing the issue**: Depending on the priority, the development team will fix the defect. High priority issues will be fixed first and low priorities last.
3. **Reporting the fix**: Once the defect has been fixed, the development team must inform the testers that the defect has been fixed and how it was fixed (e.g. changing code, altering files etc). This information will assist the testers understand the cause of the defect.

From a process point of view, all defects, no matter how they are prioritised, are identified as critical. This means that low priority defects are not necessarily less important that the high priority defects.

Minor defects can assist in preventing occurrences of similar defects in future releases/systems thus should not be disregarded. The impact of a less important defect on the system is a big deal.

Based on the defect and its origin, the changes made to the validation, prevention and documentation processes can be viewed carefully in order to catch defects earlier in the future, thus creating an overall less-expensive system.

1. Test Strategy

The time available for testing is limited and as so, everything cannot be expected to be tested with equal diligence. This means that choices must be made regarding the depth of testing. Also, it is strived to divide test capacity as effective and efficient as possible over the total test project. This principle is the basis of the test strategy.

The test strategy is based on risks. A system must function in practice to an extent that no unacceptable risks for the organization arise from it. If the delivery of a system brings along many risks, thorough testing needs to be put in place; the opposite of the spectrum is also true – 'no risk, no test'.

The first step in determining the test strategy is the execution of a product risk analyses.

The test strategy is subsequently based on the results of the risk analyses. The test strategy lays down what, how and when (in which test level) is being tested and is focused in finding the most important defects as early as possible for the lowest costs. This can be summarized as testing with an optimal use of the available capacity and time.

8.1 Test Objectives

The main objective is to verify that the Resource Manager project meets requirements and functions as stated in the specifications. Functional testing will run through each use case to identify any bugs which can be then fixed and reused as a suite before releasing any changes to continue to assure the integrity of the solution.

The goal is to refine the final version of the solution ready for release, and to create a reusable base of tests for other stages.

8.2 Product Risk Analyses

The product risks are determined in cooperation with the client and the other parties involved. This product risk analyses (PRA) is comprised of two steps:

* Make an inventory of the risks that are of interest
* Classify the risks

The acceptant has determined the product risks. The extent of the risk (the risk class) is dependent on the chance of failure (how big the chance is that it goes wrong?) and it depends on the damage for the organization if it occurs.

The risk class (RC) determines the thoroughness of the test. Risk class A is the highest risk class and C is the lowest. The test strategy is subsequently focused on covering the risks with the highest risk class as early as possible in the test project.

First the chance of failure and damage are determined for each risk. The risk class has been taken directly from this.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Business Value (Priority)** | **Damage** | **Chance of Failure** | **Result** |
| Incorrect functionality | High | High | High | A |
| Not user-friendly | Medium | Low | Low | C |
| Difficult to maintain | Low | Low | Low | C |
| Low efficiency | Medium | Medium | Low | B |
| Vulnerable security | Medium | Medium | Medium | B |

8.3 Strategy

For each risk from the product risk analysis the risk class is qualifying the thoroughness of the test. Risk class A is the highest risk class and C the lowest. The test strategy is subsequently focused on covering the risks with the highest risk class as early as possible in the test project.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Characteristic /object  part** | **PRA-RK** | **Development test** | **ST** | **FAT** | **Alpha UAT** | **Beta UAT** | **Impl** |
| Functionality | A | ⬤⬤⬤ | S | S | ⬤ | I | I |
| Part 1 -Users updating their skill profile | A | ⬤⬤⬤ | S | S | ⬤ | I | I |
| Part 2- Project managers assigning resources | A | ⬤⬤⬤ | S | S | ⬤ | I | I |
| Part 3 - Project managers create projects | A | ⬤⬤⬤ | S | S | ⬤ | I | I |
| User-friendliness | C |  |  | S |  | I | I |
| Performance | B | ⬤⬤ | S | S | ⬤ | I | I |
| - online | B | ⬤⬤ | S | S | ⬤ | I | I |
| - batch | B | ⬤⬤ | S | S | ⬤ | I | I |
| Security | B | ⬤⬤ | S | S | ⬤ | I | I |
| Maintenance | C |  |  |  |  | I |  |