**3 Test Strategy**

**3.1 Introduction**

This Test Strategy defines the processes by which all functional and non-functional testing will be

undertaken within MagicBox releases, and describes the types of testing that need to be considered,

discussed and agreed.

It documents a structured approach to testing that is based on best practice, and will communicate to

the Program and Project Management teams, and their relevant business and technical architects, how

testing will be planned, conducted, controlled and reported.

**3.2 Sprint Level Testing Workflow**

The diagram below shows the testing workflow for MagicBox product within a given sprint. The

development methodology being used is Agile Scrum. It has been decided that the feature set targeted

for a given release will be developed in one or more sprints of 2 weeks each.

The features slated to go in a particular sprint will be elaborated as User Stories and a sprint will involve

activities such as User Story Elaboration, Planning (Understanding, Estimating), High/Low Level

Designing, Coding, Identification of Test Scenarios, Test Case documentation, Identification of Risks,

Early Feedback, Test Case execution, Defect Logging, Defect Fixing, Defects Verification and Sign-Off.

**3 Test Strategy**

**3.1 Introduction**

This Test Strategy defines the processes by which all functional and non-functional testing will be

undertaken within MagicBox releases, and describes the types of testing that need to be considered,

discussed and agreed.

It documents a structured approach to testing that is based on best practice, and will communicate to

the Program and Project Management teams, and their relevant business and technical architects, how

testing will be planned, conducted, controlled and reported.

**3.2 Sprint Level Testing Workflow**

The diagram below shows the testing workflow for MagicBox product within a given sprint. The

development methodology being used is Agile Scrum. It has been decided that the feature set targeted

for a given release will be developed in one or more sprints of 2 weeks each.

The features slated to go in a particular sprint will be elaborated as User Stories and a sprint will involve

activities such as User Story Elaboration, Planning (Understanding, Estimating), High/Low Level

Designing, Coding, Identification of Test Scenarios, Test Case documentation, Identification of Risks,

Early Feedback, Test Case execution, Defect Logging, Defect Fixing, Defects Verification and Sign-Off.

As the tester becomes familiar with the user story both functionally and technically, she should be able

to provide near-accurate estimates for the effort that will be involved. The effort can be expressed in

person-days or person-hours as per the team’s convenience. Typically, the tester will be required to

provide an estimate for the following activities

o Identification of Test Scenarios

o Documentation of Test Cases

o Early Feedback

o Test Case Execution

o Defect Verification and minimal regression

The following section will illustrate how test estimation can be done and discusses a few techniques for

doing so.

**3.3.3 How to do Test Estimation?**

Estimation is an activity, which is performed in about every task that you choose to accomplish on a

daily basis. For example, *suppose, you have a client meeting at 9 AM in your office and you reside 10 KM*

*away. In order to attend this meeting you need to be in the office by about 8:50 AM. So, you start*

*planning a day before. You do an estimation of time it will take to reach office, which will eventually*

*decide the start time. You analyze that the morning traffic is usually heavy so it will take about 20 min*

*longer than the usual. You also notice that your car is running low on fuel so you need to get it refueled*

*at the nearest station, which takes about 15 min. You recall that the highway has a patch that is under*

*repair and hence slows you down by about 10 min. Also, the city’s traffic is unpredictable so it will be*

*wise to take an extra 10 min for contingency. Now, if the usual time to office is about 25 min, the*

*estimated time it will take to attend the client meeting will be the*

**Estimated Time** = Usual Time + Additional Time = 25 min + [20 min + 15 min + 10 min + 10 min] = **80 min**

Based on the above estimates, you should start from your home at about 7:30 AM to reach office by

8:50 AM.

While the above example may sound very trivial, it is in fact the basis for most estimation techniques.

 **Expert Judgment:** Using your own expert judgment is the most widely used estimation

technique. However, it must be practiced with great care and should inevitably be based on real

facts and data. As you keep working on things of similar nature, you tend to acquire a deep

knowledge of it. This knowledge is made up of an understanding of the technical complexity, the

constraints that were faced previously, the time it took to test a similar feature earlier or the

estimates that were given for a similar feature earlier. With this information handy, you are

likely to provide a good estimate of your current effort.

 **Task Breakdown:** A good estimate of effort involved in accomplishing a task can be done by

breaking the task into smaller chunks that can be estimated more readily. In our “Time to reach

office” example, we had estimated for many smaller parts of the travel rather than estimating

for the whole journey together. Same logic can be applied to the testing tasks. A task can be

broken down into several easy to estimate sub-tasks.

 **Three-Point Estimation:** This estimation technique is based around three points of estimation.

These three points are

o Best Case Scenario (B): The time it will take to accomplish the given task under best

conditions where nothing can go wrong

o Likely Scenario (L): The time it will take to accomplish the given task under normal and

likely conditions

o Worst Case Scenario (W): The time it will take to accomplish the given task under worst

conditions where anything that can go wrong will go wrong

Once the above times are handy, the three point formula can be applied to get the Estimates

(E), so that **E = B + (4 \* L) + W/6**

 **Delphi Technique:** Delphi Method is based on surveys, and basically collects the information

from participants who are experts. In this estimation technique, team members are asked to

give an estimate for the given task based on any technique that they prefer. The responses are

collected in the form of a survey. Multiple rounds are conducted till a final estimate is arrived at.

The results so obtained are quantitative as well as qualitative. This technique gives a good

confidence in estimation.

 **As a Percentage of Development Effort:** Although this technique is strongly discouraged, but

can be handy if the time to give estimates is less. As per industry trends and depending on

nature of project\feature ranging from Low, Medium to High complexity, the test effort can be

typically in the following range

**Project\Feature Complexity Testing Effort (as a %age of development Effort)**

**Simple** 30%

**Medium** 40%

**Complex** 50%

When making your estimates, it is always advisable to

 Use a mix of the above techniques. Usually two will suffice.

 Include contingency over and above the estimates. Usually 15% contingency is good enough.

**3.3.4 Identify High Risk Areas**

Within the planning phase, as the user stories become clear to the team and better effort estimates are

ready to be made, it is important for tester to identify and highlight the risks that are underlying. A

timely identification of risks can help scrum master and team to work on a mitigation plan. The risks

likely to exist can be under the following heads

 **Inherent schedule flaws:** Since, it is not possible to give a 100% correct test/development effort

estimate

 **Requirement dynamics and inflation:** Since requirements keep changing and new ones keep

emerging

 **Employee Turnover:** Since people can leave the projects midway or go on leaves

 **Unclear Specifications:** Since requirements may not be fully understood at the time of planning

**3.4 Test Preparation**

From testing standpoint this is the next phase that follows the planning phase in a given sprint. This is

where a tester prepares her for the test execution to come. This is also where the feature(s) take shape

as HLD, LLD and coding goes on in parallel. The tasks included are as follows

**3.4.1 Identify Test Scenarios**

Identification of test scenarios is when the tester makes key notes about the system under test and

looks at it from different angles and perspectives. Tester tries to analyze various ways in which a user is

likely to interpret and use the feature exploring both positive and negative scenarios. It does not require

any documentation other than making single line notes or drawing a map of various test points. Test

scenarios become the basis for the more formal test cases

**3.4.2 Write Test Cases**

In line with agile principles, we do not want to emphasize on detailed documentation of test cases with

elaborate steps and templates. It is understood that the tester is capable of performing the basic steps

needed to run the test case without having to refer to a document. However, some basic information

needs to be recorded in order to reuse it in future, to make sure that good test coverage has been given

and to enable automation of these test cases in future.

Test Cases are the logic of what needs is to be tested and the expected result of that test. They do not

need the data on which they will be executed to be defined, unless it is relevant to the expected result,

nor do they require detailed steps to define the exact set of actions to execute them.

Using this rationale a Test Case can be written as

**GIVEN** <pre-condition>

**WHEN** <some event takes place> or **WHEN** <cause> or **WHEN** <condition>

**THEN** <some expected result> or **THEN** <effect>

Test Cases must be recorded within JIRA and must have a mapping with user stories.

**3.4.3 Provide Early Feedback to Developer**

This task has been included in the preparation phase as a means to provide early feedback to the

developer who is coding for the given user story. This may be understood along the same lines as TDD.

The objective is to “Fail Fast” so that the steps for recovery are easier and cost-effective. More details

on this have been provided under section 2.4.

**3.5 Test Execution**

**3.5.1 Execute Test Cases on Staging Environment**

The build containing features to be tested will be deployed to the Staging environment. Testers will have

access to the Staging environment and execute their test cases on this environment. Any failures will be

reported as defects in JIRA.

The build to be tested must be available for testing according to the planned schedule. If this time limit

is not met, then there will be a cascading effect on the test estimates and consequently testers will

continue testing beyond the agreed upon timelines that could result in slippage of release schedule.

Test execution may also be affected by the number of defects filed on staging environment. In case a

very high number of defects are found, the testing schedule will slip. However, with the introduction of

Early Feedback, an unusually high number of defects are not anticipated.

**3.5.2 Mark As Pass**

Test Cases that confirm the correct implementation of functionality will be marked as Pass and same will

be recorded in JIRA.

**3.5.3 Mark As Fail and Log Defect**

Test Cases that confirm the incorrect or erroneous implementation of functionality will be marked as

Fail and same will be recorded in JIRA.

Moreover, a defect will be logged in JIRA corresponding to the failed test case. A typical defect will have

the following information

**Defect Summary:** A one line statement that gives a short description of the defect

**Description:** A detailed description of the defect that provides Steps to Reproduce, Actual result,

Expected Result, Environment details (OS, Browser, Server), Any other critical information that can

enable a quick fix for the defect

**Attachments:** A tester is expected to attach evidence in support of the defect including but not limited

to Screenshots, Logs, Reports (especially in case of performance defects) and Videos (if needed)

**Severity:** Severity is the measure of the impact that the given defect has on the system under test. The

following table describes the severity level and its meaning

**Severity Meaning**

**Blocker (S1)**

Complete absence of the feature or functionality to be tested or failure

leading to complete system shutdown

**Critical (S2)**

The defect affects critical functionality or critical data. It does not have a

workaround. Example: Unsuccessful installation, complete failure of a

feature

**Major (S3)**

The defect affects major functionality or major data. It has a

workaround but is not obvious and is difficult. Example: A feature is not

functional from one module but the task is doable if 10 complicated

indirect steps are followed in another module/s

**Minor (S4)**

The defect affects minor functionality or non-critical data. It has an easy

workaround. Example: A minor feature that is not functional in one

module but the same task is easily doable from another module

**Trivial (S5)**

The defect does not affect functionality or data. It does not even need a

workaround. It does not impact productivity or efficiency. It is merely

an inconvenience. Example: Petty layout discrepancies,

spelling/grammatical errors

**Priority:** Priority is the measure of the impact that the given defect will have on the business. It is used

to describe the timeframe in which the defect needs to be fixed. Priority can be initially set by the tester

but it must be validated and changed if needed by the product owner or the project manager. Following

table describes the priority level and its meaning along with the time in which defect of that priority

level needs to be fixed.

**Priority Meaning**

**Critical (P1)**

The fix for this defect is the highest priority for the business. It must be

fixed within 8 hours or the same day and may need an interim build

that contains the fix.

**High (P2)**

The fix for this defect has a high priority but development work can

proceed normally and include a fix for this defect in the next scheduled

build. Usually it can be fixed in 24 hour time frame.

**Medium (P3)**

The fix for this defect will be pushed in only if there are no P1 or P2

defects to be fixed. However, teams must make sure that all P3 defects

are resolved in the final released build.

**Low (P4)**

The fix for this defect is low on priority. Teams may mutually decide to

not fix this defect at all if the cost to fix is high. Usually, the released

build will have one or more instance of known P4 defects.

**3.5.4 Verify Defect Fix**

As and when the development team checks in the fix for a given defect and updates the “Resolution” of

the defect to “Fixed”, the tester will verify that the defect was actually fixed or not. Defects will be

verified only on staging environment before being closed if the fix has actually been done or reopened if

the bug still persists. Results will be accordingly marked in JIRA.

Tester will also perform minimal regression testing while verifying a defect.

**3.6 Quality Gates – Entry and Exit Criteria**

**3.6.1 Exit Criteria**

Within a given sprint, the next phase will only start if the exit criteria of the preceding phase have been

met at the Quality Gate.

**Planning Phase:**

 All User Stories were covered in the review

 QC team understands all user stories that will be part of the sprint

 QC team was able to provide an effort estimate for all testing tasks within all user stories that

will be developed in the sprint

 Either all risk items were called out and recorded along with mitigation plan or it was mutually

decided that there are no risks anticipated by the team.

**Test Preparation Phase:**

 Tester was able to identify all possible test scenarios for the given user stories

 Tester was able to write test cases against all test scenarios

 Tester was able to provide an early feedback for all user stories that would be developed in the

current sprint

 All features and functions were successfully developed in line with early feedback and a build

was pushed to Staging for further testing

**Test Execution Phase:**

 Tester was able to execute all test cases for the given user story

 All valid defects were accurately logged within JIRA

 All fixes were verified by the tester and status accordingly set in JIRA

 All Blocker, Critical and Major severity defects were closed

 All Critical and High priority defects were closed

 There are no Medium priority defects left in the system but with a mutually agreed acceptance

and plan for resolution

**3.6.2 Entry Criteria**

Each phase will only start if the following criteria have been met

 There must be a record in the form of a simple note or comment or a detailed document or an

email communication or status changes marked in JIRA to validate that the preceding phase has

completed.

 The “exit criteria” of the previous phase must have been satisfied

**3.7 QC Sign-Off**

QC Sign-off at the end of a given sprint will be based on

 Successful development of all features\User Stories that were planned for the given sprint.

 Successful execution of all test cases planned for the given sprint or in case where there is

schedule slippage, it may be changed to execution of all high priority test cases. This will be

decided mutually between team, scrum master and product owner. Functional managers may

be asked for opinion as well.

 Successful closure of all Blocker, Critical and Major Defects, except where a known defect can be

carried into the next sprint based on mutual agreement between team, scrum master and

product owner. Functional managers may be asked for opinion as well.

**5 Testing Types**

The types of testing that will be undertaken by the MagicBox testing team can be discussed under two

separate headings, **Functional Testing** and **Non-Functional Testing**. Both types of testing will have a

well-defined scope and will be undertaken in accordance with an agreed upon plan within a sprint.

**5.1 Functional Testing**

Testing for all stated requirements that define the partial and/or overall functionality of the system will

be a part of Functional Testing. The requirements will be made clear to the tester in the form of welldefined

user stories recorded in JIRA. As already elaborated in section 3.3, the tester will be part of the

sprint planning where the user stories will be discussed and understood. Based on this understanding,

tester will prepare and execute functional test cases.

Broadly, functional testing will be covered under the following types

**5.1.1 Sanity Testing**

Tester will reserve a minimal number of functional test cases that test the basic flows and functionality

of the application under test. The objective of Sanity testing will be to prove that the application is

acceptable for further testing. The sanity tests will be run every time, a build is provided to testers on

Staging server. Exception to this can be decided by the team and mutually agreed upon between tester,

developer and scrum master. It is highly recommended to run Sanity tests when a build contains many

new features or bug fixes. Testing team will plan to automate sanity test cases to save execution time.

**5.1.2 Feature Testing**

Tester will spend most time on proving the functionality of the features to be released within a sprint.

These tests will be based on the user stories that have been selected for the sprint. The objective of

features testing will be to test the features for all possible scenarios from user and technical point of

view. It will cover all positive and negative scenarios. The test cases will be peer reviewed (within scrum

team) before being finalized. The test cases will also be prioritized based on the risk posed by the failure

of the given feature. A high risk feature will correspond to a high priority test case. Test cases will be

executed in order of priority to prove the high risk items earlier in the cycle. It is mandatory for tester to

execute all feature tests before closing the test cycle. Some feature tests may become part of the Sanity

test suite and all feature tests will become part of the regression suite.

**5.1.3 Regression Testing**

Regression test cases aim at uncovering the defects that are introduced in one part of the system as a

result of defect fixing and/or new code written in another part of the system. All feature test cases that

have undergone the cycle of feature testing will become part of the regression test suite. If a feature

itself changes over a period of time, the regression test suite will be updated to make sure that the tests

are valid for the current feature set.

There are two ways in which team will run the regression test suite or part of it.

 **As part of the Sprint with new features:** Where a given sprint has new features under

development, a certain part of the sprint (1 – 2 days) maybe reserved for performing regression

tests. This may or may not cover the entire regression test suite. Team will pick the most

relevant tests to be run from regression test suite. Testing will be done on Staging environment.

 **As part of a separate Sprint:** Regression tests can be run as part of a Sprint (1 – 2 weeks)

dedicated only for regression testing. This Sprint will not have any development effort and no

new features will be rolled out. Such a sprint is recommended prior to any major release.

Testing will be done on Staging environment.

**5.1.4 User Acceptance Testing**

The objective of User Acceptance Testing (UAT) is to test that the product is ready to be accepted by the

end user. The scope will vary depending on the amount of time and resources available. Test cases to be

run will mainly focus on end-to-end workflows of basic type plus the most widely used features and

functionality.

**Pre-Production Environment**

Following high level scenarios will be covered on the Pre-Production environment. Please note that the

coverage in each scenario will vary in detail depending on the availability of time and resources

**S. No. Test Scenario Area**

**1.** Abcd Platform

**2.** Pqrs Device

**3.** Lmno Assessment

**Production Environment**

UAT on production environment will comprise of testing scenarios for various tenants for whom the

release is being made. Testing team will own a checklist of scenarios that should be covered. A

dedicated resource will run through the checklist along with the implementation team. The checklist to

be covered has been uploaded here on Basecamp. The same checklist on Google Drive can be found

here.

**5.1.5 Environment Specific Testing**

The purpose of Environment Specific Testing is to make sure progressively, that our product remains

compliant with the peripheral software/hardware that it needs to work on. For example, our product

works on a variety of OS such as Windows (7, 8, XP), MAC, Android, iOS and supports a variety of

browsers such as IE, Firefox, Chrome, Safari, Native device browsers. As a testing team, we need to be

abreast with changing trends and features in these peripherals. In case we come across any change that

might affect our product, then we must undertake the following actions

 Ascertain the functionality that will be affected

 Gauge the impact on end user

 Bring it to the notice of PM, test/dev leads for further assessment

 Log an issue/ticket in JIRA for tracking purpose

However, to achieve the above, team needs to be aware of how our partners/peripherals are

progressing. We need to keep our eyes open to notice any changes and keep ourselves updated with the

announcements, press-releases and roadmap of these software partners. Each team member will take

out 1-2 hours per week to identify such changes and bring it up for discussion.

**6 Overall Testing Lifecycle**

**Release Planning:** At this stage, the prod

release. The scope for the release will be frozen at this level.

from 1 day to 1 week depending on the scope.

**Feature Development:** At this stage the ne

development and early feedback happens on DEV environment. Testing will be done on STAGING

environment. QC will sign-off before code freeze as per section 3.2.

depending on over all scope and feature complexity.

**Regression Testing:** Once code freeze is announced, the testing team will perform regression test on

STAGING environment. It is recommended to make use of automa

team may take one full week to perform regression testing that will result in at least 90% confidence in

the system. For more details, please refer to section

**ifecycle**

**User Acceptance Testing:** Testing team will perform UAT on the Pre-Production environment. The

objective will be to make sure that all basic workflows work flawlessly. Depending on the time available

in the release, testing team will undertake 1 day, 3 days or 5 days of UAT. For details please refer section

**7 Test Management and Governance**

**7.1 Defect Management**

JIRA will be used to file all defects found in any feature/module/functionality of MagicBox application.

Defects will be linked to the user story against which they have been filed. Testers, developers, product

owners and leads will review, update defects within JIRA. The person who has initially logged the defect

will be the owner of that defect and responsible for its closure after fixing. For details on Defect filing

process, please refer to section to **3.5.3**.

**7.2 Defect Triage**

Individual scrum teams will engage in defect triage at regular intervals. The objective of defect triage will

be to filter out the defects for fixing, re-prioritization or changing the status of a defect based on review

and discussions during the triage. Defect assignment and/or ownership can also be changed after this

discussion.

On need basis, a defect triage can be done for defects from more than one scrum teams and cal for

participation from product management and functional management.

**7.3 Test Case Management**

**7.4 Test Reporting & Metrics**