22/09/2021

Amai Rana, Ashley Bending

The software institute

BlackJack

Test Plan

Contents

[Revision History 1](#_Toc82591741)

[0.0 – Introduction 2](#_Toc82591742)

[1.0 - Test Items 2](#_Toc82591743)

[1.1 - Features to be tested 2](#_Toc82591744)

[1.2 - Feature no tot be tested 3](#_Toc82591745)

[2.0 – Approach 3](#_Toc82591746)

[2.1 - Tools used 3](#_Toc82591747)

[2.2 - Configurations 4](#_Toc82591748)

[2.3 - Component testing 6](#_Toc82591749)

[2.4 - Acceptance testing 6](#_Toc82591750)

[3.0 - Item Pass/Fail Criteria 7](#_Toc82591751)

[3.1 - Suspension and resumption criteria 7](#_Toc82591752)

[4.0 - Test Deliverables 7](#_Toc82591753)

[5.0 - Test Tasks 8](#_Toc82591754)

[5.1 - Responsibilites 9](#_Toc82591755)

[5.2 - Approvals 9](#_Toc82591756)

# Revision History

|  |  |
| --- | --- |
| Revision History: | 01 - 15/09/21 |
|  | 02 – 20/09/21 |
|  | 03 – 22/09/21 |

# – Introduction

The Software Institute has requested as part of our training to design and build a version of the card game Blackjack. This will be produced in Java, and will be utilising additional features such as Front/back end Microservices Testing. The project as a whole will be produced in a TDD (Test Driven Development) and BDD (Behaviour Driven Development) format. Using this method means that the tests and requirements will be developed first and then tested, subsequently failing and then producing the code to fulfil the previous tests and requirements.

# - Test Items

This section of the plan involves introducing and discussing the items that would be tested. This area is written from a testing perspective and as such may include technical language.

## 1.1 - Features to be tested

* Player needs to initialise game
* 2 cards should be seen placed in the player’s deck
* Dealer should have 2 cards which can be seen by the player
* Player should be able to perform a ‘hit’ action to add a card and see in their deck
* Player should be able to perform a ‘stay’ action to end their turn, starting the dealers
* Player should be able to place a bet
* Player should be able to win by either dealer going ‘bust’, or player cards totalling 21
* Player should be able to lose by going ‘bust’ or dealer cards being higher than player
* Player should not encounter cards above 10, or below 1

Current tests:

* See if new deck is created
* See if amount in cards is equal to 52
* See if amount of cards in each suit is equal to 13
* Check suits are the correct values for a deck of cards
* See if a new card is created
* See if a new card has the correct values after being created
* See if a new card has the correct suit after being created
* See if a new stack of cards can be created
* See if a deck contains 4 suits
* See if a deck contains 13 cards of diamonds
* See if a deck contains 13 cards of hearts
* See if a deck contains 13 cards of spades
* See if a deck contains 13 cards of clubs
* See if a new deck contains the correct values
* See if a deck has the game rules properly applied
* Testing what happens if player disconnects from network during a data transfer

## 1.2 - Features not to be tested

* Player should not be able to add any unexpected inputs
  + Most player inputs will be provided in the code. Upper boundary limits of any inputs will be tested to ensure system is secure

# – Approach

## - Tools used

|  |  |  |
| --- | --- | --- |
| **Tool** | **Use case** | **Training required** |
| IntelliJ Idea Community | IDE used for creation of code and tests | General knowledge of Java required |
| Visual Studio Code | IDE used for code editing. Primarily used for build/debugging web and cloud applications | General knowledge of Java and Javascript required |
| Maven | Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a projects build, reporting and documentation from a central piece of information | Tool setup can be found in the documentation. Basics of working with XML files will be required. |
| JUnit | Unit testing framework for Java programming language | An extended knowledge of coding and the basics of test structure required. |
| Jenkins | CI/CD tool that automates software building, testing and deployment | A knowledge of software development processes, Git and Maven. |
| SonarCloud | SonarCloud is a cloud-based code quality and security service. | How to setup this tool can be found in the documentation, basics of working with XML files will be required |
| Git | A command line interface used for interacting with repositories | A basic knowledge of git commands is recommended |
| GitHub | A cloud hosted repository | Basic knowledge of how cloud-based repositories is required |
| GitHub Actions | Used for building the application within the cloud | A knowledge of how to work with .yml files. |
| Jira | A Kanban method of agile project management | Basic knowledge of Kanban and scum methodology |
| Selenium | A web framework that allows for the execution of cross-browser tests. Can be used to automate manual testing. | Base knowledge of Junit and maven required |
| Springboot | An open source micro framewrork that embeds Tomcat into the project for running applications | Base knowledge of Java and Maven is required |
| React | A javascript library for building user interfaces in the form of web applications. | A basic knowledge of Javascript is required |
| Amazon Web Services |  |  |
| JaCoCo | Jaccoco is an open-source toolkit for measuring code coverage in a code base and reporting it through visual reports | How to setup this tool can be found in the documentation, basics of working with XML files will be required |
| Jest | Open source framework built to perform unit tests on React based web applications |  |
| Material -Ui | A library that allows the importing of components to create a user interface in the React applications. | Basic knowledge of react required |
| TestNG | A testing framework that expands upon JUnit | Basic knowledge of Java required |
| Cucumber/Gherkin | Helps perform acceptance testing via user stories produces by Cucumber/Gherkin | Basic knowledge of testing and Java required |

## – Configurations

This section includes the configurations of the systems that the tests will be run on.

It includes:

* The hardware specifications of the development and test machine
* The version of the local software used
* The dependencies and versions
* Any extra configurations done regarding quality gates and other build settings

**Hardware:**

**Development Environment 1:**

* OS: MS Windows 10 Pro – Build 19043
* CPU: i9-9900kf
* RAM:32gb ddr4
* Storage: 250GB NVMe SSD, 500GB NVMe, 1TB Sata SSD
* GPU: Nvidia RTX 3080
* IDE: IntelliJ IDEA Community 2021.2
* IDE: Visual Studio Code

**Development Environment 2:**

* OS: MS Windows 10 Home Build 19042
* CPU: Intel Core i7-8750H @ 2.20GHz
* RAML 8GB DDR4
* Storage: 1TB HDD, 256GB SSD
* GPU: Nvidia GeForce GTX 1050 Ti
* IDE: IntelliJ IDEA Community 2021.2

**Software:**

* IntelliJ IDEA Community – Version: 2021.2.1
* Visual Studio Code

**Dependencies:**

*Maven Plugins and dependencies:*

* Apache-Maven – Version: 3.8.2
* Maven-Surefire plugin: 3.0.0-M5
* Jenkins-Maven plugin
* Jacoco-Maven plugin: 0.8.7
* Springboot-Maven plugin: 2.5.4

*Junit Plugins and dependencies:*

* Jupiter API- 5.7.2
* Jupiter Params- 5.7.2
* Jupiter Engine- 5.7.2
* Cucumber-Java- 6.11.0
* Cucumber-Junit-6.11.0

*Json conversion dependencies*

* Gson – 2.8.5

**Cloud Software:**

* SonarCloud

*Extra build configuration tasks:*

* SonarCloud key must be provided to GitHub actions before using secrets method
* Target build environment: Ubuntu-Latest
* Sonar Org: Asharubendy

*Quality Gate configuration tasks:*

* Counter: COMPLEXITY
* Value: COVERED RATIO
* Minimum: 0.8 (80%)

**Metrics to be collected:**

* Security issues
* Potential vulnerabilities
* Security Hotspots
* Maintainability
* Code Smells
* Debt
* Statement Coverage
* Test Coverage
* Amount of tests
* Duplications
* Test competition

## - Component testing

***Functional tests:***

This involves testing functions of the software. An example of this would be Assertion based tests.

In context, one of the tests included in the test items is the assertions that the function that returns ‘-1’ when the players total hand value is above 21, which indicates the prerequisites for a loss condition, and would actually return ‘-1’. This in turn would provide a True or False value within the test; these tests are measured by Junit and there will be a pass/ fail criterion.

***Non-Functional tests:***

This involves testing non-functional aspects of the software application. An example of this would be Load based tests.

The aim of this would be to examine what would happen to the system performance in normal and peak conditions. To do this, multiple applications of the project would be run, subjecting the server to a lot of traffic, until there was a significant change in the system performance.

***White box tests:***

Tests which are designed to achieve complete statement and decision coverage. This is measured by the metric of Test and statement coverage from the SonarCloud and Jacoco report.

## - Acceptance testing

*User acceptance testing*

User acceptance testing is performed through the use of user stories performed through Cucumber and Gherkin.

*These are written in the Gherkin language and performed in this format:*

**Scenario –** (In context) The player can place bets

**Given –** The game is open

**When –** I am on the betting screen

**Then –** I should have the option to place a bet

**Significant Constraints:** Provision of additional elements to the project.

As the project is in development and is using the TDD (Test Driven Development) process, there is always the potential for more tests to be added. This case is made further certain by the introduction of microservices, web elements and UI elements being added to the project.

**Deadline:** These should be completed by 23/09/2021, or when Gareth decides to stop the project.

# - Item Pass/Fail Criteria

There are multiple Pass/Fail criteria for each test as well as an overarching Pass/Fail criteria.

The first of these is an initial quality gate, this prevents the build form compiling if the code’s overall coverage is below 80%. It checks against Jacoco’s generated report, which tests how much of the code is covered by a test.

The next step is the pass / fail criteria of the functional and non-functional tests, 100% of these tests need to be passed in order for it to successfully build into GitHub actions. The result of this will be shown in SonarCloud.

## - Suspension and resumption criteria

There are a number of reason that could occur for which testing would have to be suspended, a few examples of this could be:

* Critical software defects
* Massive defect backlogs
* Testers falling ill
* Testers having to take unexpected time off work

These can be remediated through waiting for testers to return.

* Waiting for developers to fix the defects found
* Waiting for developers to clear the backlogs.

# - Test Deliverables

There are a number of deliverables that will be expected at the end of the test plan, these are:

1. **The Test plan**
   1. The test plan document and its associated deliverables
2. **The Test environment**
   1. The test environments have been set up, this includes both the hardware and software
3. **The Test suite(s)**
   1. Collections of tests that are intended to be used to test the software
4. **The Test stubs**
   1. Objects that contain data that will be used to feed information to the tests.
5. **Defect reports**
   1. Reports that signify a defect that has been detected. This includes information such as date and time, the line or method in the code, the steps taken to replicated it in order to make it as easy as possible to find and fix.
6. **Results**
   1. The final results of the tests themselves
7. **Test completion / evaluation**
   1. Finalising the test and defect reports for the stakeholders.

# - Test Tasks

A test process consists of the following main groups and activities:

1. Test planning
   1. Defining the objectives of testing and the approach for meeting test objectives within constraints such as specifying suitable test techniques
2. Test monitoring and control
3. Test analysis
   1. Analysing the test basis in regard to requirements and user stories
   2. Assessing the test quality in terms of clarity and consistency
4. Test design
   1. Designing how the tests run for each feature to be set
   2. Designing the test environment in terms of software and hardware requirements
   3. Designing the tests to fail as software not designed for it yet
5. Test implementation
   1. Creating test suites
   2. Prioritizing test procedures
   3. Arranging test suites within a test execution schedule
   4. Building the test environment
   5. Preparing the test data
6. Test execution
   1. Executing the tests
   2. Comparing results with expected results
   3. Defect reporting
   4. Developing software to pass the test
   5. Repeating of test activities as part of regression testing
7. Test completion
   1. Checking defect reports are closed
   2. Creating a test summary report
   3. Consulting other team members and stakeholders on information gained.

## – Responsibilites

There are responsibilities that are attached to certain roles within the test plan. Since we have been tasked with producing the software, test plan and its execution. As a result, this additionally makes us the test manager and tester. Gareth is considered the project manager, providing the requirements and feedback.

|  |  |  |
| --- | --- | --- |
| **Role** | Person(s) | Included responsibilities |
| Developer, Tester, Test manager | Ashley Bending | * Write and update test plans * Write and update documentation * Complete tests, from analysis to completion * Complete all tasks |
| Amai Rana |
| Project manager | Gareth Davis | * Provide product requirements * Provide technical training * Provide feedback on test plans and product |

## – Risks and contingencies

Some of the potential risks of this test process are:

* Changes to testing requirements will need additional testing
* Testing team may require extra time to complete all tests
* Development of fixes may not work, leaving errors unresolved

## 5.2 - Approvals

Initial approval of the test results will come from Amai and Ashley. In the event that the project meets the specifications, it would be classed as a success. Final approval will be retrieved from Gareth, the project manager, determining if the project has met the specifications and that it is of a high standard.