



**Faculty of Engineering & Applied Science**

**SOF3980U – Software Quality**

**Assignment 1 - Hangman**

**Due Date: 02/17/2022**

<b>First Name</b>	<b>Last Name</b>	<b>Student ID</b>
Abdul	Bhutta	100785884

## **1.1 Introduction**

In this application, the stakeholder/customer is Abdul Bhutta. The purpose of having a stakeholder is to view the application from a different perspective and have requirements to work with. As the customer, I would like to see an application that is designed to simulate the game of hangman. The application was designed and implemented using eclipse to write the code using the JAVA language and the testing was done with junit. The screenshots of the application and test cases can be seen in the appendix below. The application was run on the eclipse console. The game is simple where the player is given a random word from the dictionary and is asked to guess the word. The user will be prompted to enter the character and if it is correct, it will display the character on the screen with the other characters hidden with the dash symbol. After the first input from the user, they will be prompted whether they would like to guess the word. If the user chooses yes, they will have a chance to guess the word and end the game or if they choose no, they will be asked to enter another character. The user will be given 6 guesses to guess the word, if the user does not guess the correct word after 6 tries, the game will end. Those are the requirements for the application and additional features may be implemented in the future as the project progresses. For now, the stakeholder wishes to view a prototype of the application.

## **1.2 Software Process**

The goal was to create a prototype of the application hangman. The approach that was used in the software engineering process was the evolutionary and agile methods. The agile approach was used to create simple features such as getting the word from an array and testing the method before moving on to the next iteration. As the stakeholder for the application, it was easy to adapt as the stakeholder was always involved in each stage. Another benefit of the agile process was the fact that a large team was not required and refactoring of the code between each iteration was done throughout the project. The evolutionary method was used to get analysis and design from the stakeholder while making sure a prototype of the application was created and making sure each stage released a functional requirement for the overall project. The first few stages of the application were implemented from the user requirements while having an impression of the application. The first prototype application was created, the next prototype/version can begin as extra features will be added in the future. This is extremely time-consuming but the benefit of displaying a prototype can help the software development team to predict whether they are on the right path while getting feedback from the stakeholders. The next prototype will be built on this with additional features and more feedback from the stakeholder.

### **1.3 The functionality of the code**

The application was designed using user requirements for the application hangman. The main method in the application was designed for user input and determining whether the game was over or won by the user. Each method/function in the code had a task/function to execute. When designing the code, the best practice is to make each function execute a single task while having less dependency on other functions allowing loose coupling. The methods implemented in the application are shown below with the functionality of the method.

<b>Method</b>	<b>Functionality</b>
<b>getWord()</b>	The method will get one word to guess for the user from a static array.
<b>printMan()</b>	It was designed to implement the hangman figure on the screen. It would take the current wrong guesses the user was currently at and would print the corresponding figure.
<b>playerGuess()</b>	The method will return true or false whether the character the user inputted was contained in the word.
<b>wordState()</b>	The method will print out the correct character at the position in the word. It will return true once the correct word length matches the user guessed characters.
<b>guessWord()</b>	The method will return true if the guessed word was true.

## **1.4 Tests**

In each iteration, a function/method for the application was designed and implemented. Once satisfied with the function, each function was tested before proceeding to the next iteration. A few test cases were not added to the application due to time restrictions and were not performing correctly, therefore removed from the final project. The test cases that were not included were to check after 6 guesses, the user was not able to input any characters. A test case to check whether the guess counter does not increase when the user inputs the correct character, and a test case to check the guess counter is being incremented right after the user inputs the wrong character. The test cases that were implemented are shown below in the table.

<b>Test</b>	<b>Test function</b>	<b>Functionality</b>
1	<b>checkWord()</b>	The test will checks the method <code>getWord()</code> and if the array for the words is not empty.
2	<b>correctWordGuessTest</b>	The test will check whether the function <code>wordstate()</code> is returning true when the correct number of characters is the same as the word.
3	<b>incorrectWordGuessTest()</b>	The test will check the function of <code>wordstate()</code> to return false when the correct guesses are not equal to the word.
4	<b>playerCharacterGuessTest ()</b>	The test checks the functionality of the method <code>playerguess()</code> where it expects to receive true if the character entered is in the word.
5	<b>guessWordTest()</b>	The test will check if the method <code>guessWord</code> is return a 1 or 0. In this test, we would like to return a 1 which means the user has guessed the word right.
6	<b>gameOverTest() {</b>	The test case will check whether the game is over. The test case should expect 1 if the game is over.

## **1.5 Issues**

One major issue in testing was the inexperience developing and testing in jUnit, there are many various ways you can test the quality of each function but for this project, due to time restriction, 6 test cases were taken. In the future, there may be more than one test for each function to provide the best performance and efficiency for the application, although I believe the quality of the application was great as the coverage for the methods was 100 percent which is shown in the appendix (2.2.7). Another issue that occurred during the testing phase was not all the tests were automated. Initially, the first method was created asked the user to enter an input this caused an issue for automated testing. The tests should always be automated and should not ask the user to input then test the case. One way I was looking to approach this issue was to simulate the user input in the test case, allowing the tests to be run without asking the user to input. Eventually, I decided the best choice was to make each method do one task for the best efficiency and performance, therefore taking out any inputs required from the method and requesting the input from the user from the main method.

## 2.0 Appendix

### 2.1 – Application

```
Hangman [Java Application] /Applications/
Hangman
Welcome!
=====
|
----
Please enter a letter:
```

#### 2.1.1 Game ending with a user entering the correct word

```
Problems Javadoc Declaration Console Coverage
<terminated> Hangman [Java Application] /Applications/Eclipse.app/Contents/Eclipse
Hangman
Welcome!
=====
|
----
Please enter a letter: a
=====
|
a----
Would you like to guess the word? no
=====
|
a----
Please enter a letter: b
=====
|
ab---
Would you like to guess the word? no
=====
|
ab---
Please enter a letter: u
=====
|
ab-u-
Would you like to guess the word? yes
Guess the word: abdul
Congrats! You guessed the word right!
```

## 2.1.2 Users win by guessing all the right characters

```
Problems Javadoc Declaration Console Coverage
<terminated> Hangman [Java Application] /Applications/Eclipse.app/Contents/Eclipse
Hangman
Welcome!
=====
|
-----
Please enter a letter: u
=====
|
u-----
Would you like to guess the word? s
Please enter only yes or no: no
=====
|
u-----
Please enter a letter: m
=====
|
u-m---
Would you like to guess the word? no
=====
|
u-m---
Please enter a letter: s
=====
|
usm---
Would you like to guess the word? no
=====
|
usm---
Please enter a letter: a
=====
|
usma-
Would you like to guess the word? no
=====
|
usma-
Please enter a letter: n
=====
|
usman
Would you like to guess the word? no
|
=====
|
usman
Congratulation you won!!
```



### 2.1.3 User lost after 6 failed guesses

```
Problems Javadoc Declaration Console
<terminated> Hangman [Java Application] /Applications/Eclipse.a

  |
  o
  \
-----
Please enter a letter: r
=====
  |
  o
  \ /
-----
Would you like to guess the word? no
=====
  |
  o
  \ /
-----
Please enter a letter: r
=====
  |
  o
  \ /
  |
-----
Would you like to guess the word? no
=====
  |
  o
  \ /
  |
-----
Please enter a letter: r
=====
  |
  o
  \ /
  |
  /
-----
Would you like to guess the word? no
=====
  |
  o
  \ /
  |
  /
-----
Please enter a letter: r
=====
  |
  o
  \ /
  |
  / \
-----
Would you like to guess the word? no
=====
  |
  o
  \ /
  |
  / \
-----
You have lost
```

## 2.2 – Testing

2.2.1 – Test to check if the array is empty, will return true if elements are in the array.

```
@Test
void checkWord() {
    assertNotNull(Hangman.getWord());
}
```

2.2.2 – Test to check if the function is working when the correct characters are inputted, will return true if all the characters guessed are correct.

```
@Test
void correctWordGuessTest() {
    String guessedWord = "test";
    String actualWord = "test";
    List<Character> playerGuess = new ArrayList<Character>();

    for (int i = 0; i < guessedWord.length(); i++) {
        playerGuess.add(guessedWord.charAt(i));
    }

    assertTrue(Hangman.wordState(actualWord, playerGuess));
}
```

2.2.3 – Test to check the method does not proceed to return true when all the characters have not been entered

```
@Test
void incorrectWordGuessTest() {
    String guessedWord = "test czse";
    String actualWord = "test case";
    List<Character> playerGuess = new ArrayList<Character>();

    for (int i = 0; i < guessedWord.length(); i++) {
        playerGuess.add(guessedWord.charAt(i));
    }

    assertFalse(Hangman.wordState(actualWord, playerGuess));
}
```

2.2.4 – Tests to check if the character entered matches the character in the word, it will return true if the entered character matches.

```
@Test
void playerCharacterGuessTest() {
    String word = "test case";
    String guess = "t";
    List<Character> playerGuesses = new ArrayList<Character>();

    assertTrue(Hangman.playerGuess(guess, word, playerGuesses));
}
```

2.2.5 – Tests to check if the guessed word was correct, if correct it will return true.

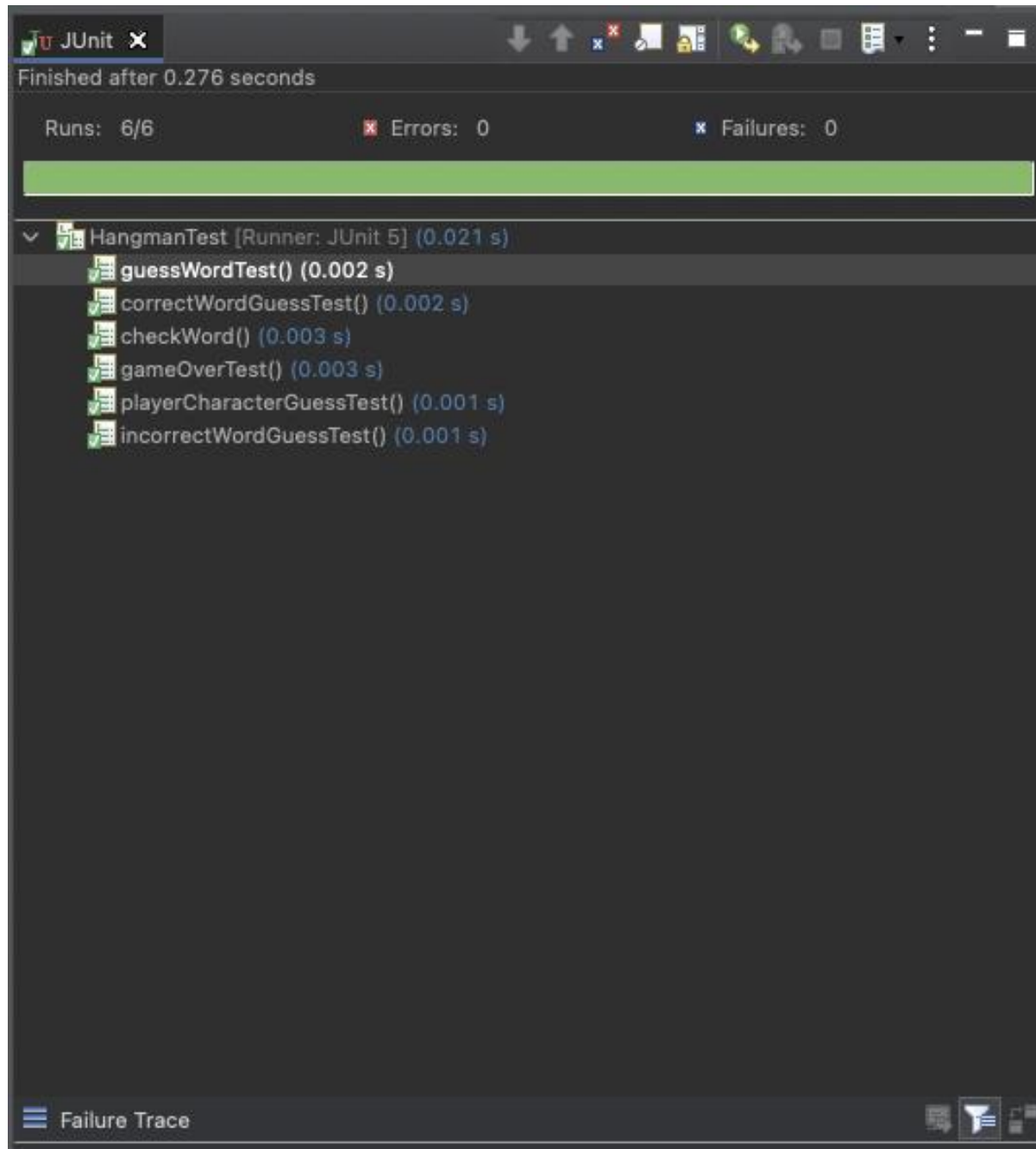
```
@Test
void guessWordTest() {
    String guessedWord = "test";
    String actualWord = "test";

    assertTrue (Hangman.guessWord(guessedWord, actualWord));
}
```







2.2.6 – Tests to check whether the game is over. Should get 1 if the game is over and 0 if not.

```
@Test
void gameOverTest() {
    assertEquals(Hangman.printMan(6), 1);
}
```

## 2.2.6 – Tests Results



## 2.2.7 – Coverage Results

 getWord()		100.0 %	48	0	48
 guessWord(String, String)		100.0 %	8	0	8
 playerGuess(String, String, List...		100.0 %	15	0	15
 printMan(int)		100.0 %	50	0	50
 wordState(String, List<Character>)		100.0 %	37	0	37
▼  HangmanTest.java		100.0 %	99	0	99
>  HangmanTest		100.0 %	99	0	99