



Software Engineering: Process and Tools

Software Unit Testing Report

Semester 2, 2020

Kieu Nguyen
Student ID: 101354326

Table of Contents

1. Executive Summary	2
2. Context and Background	3
2.1 Introduction of Testing Driven Development	3
2.2 Hangman Game.....	4
2.3 Report Introduction.....	4
3. Design.....	5
3.1 Requirements.....	5
3.2 Design and Approach.....	6
4. Reference	12

1. Executive Summary

This paper is a report for an assignment in PRT582- Software Engineering: Process and Tools to present how a small game Hangman is developed by applying Testing Driven Development (TDD). This paper will overview TDD process, the game hangman requirements and then how TDD is implemented to write the hangman program.

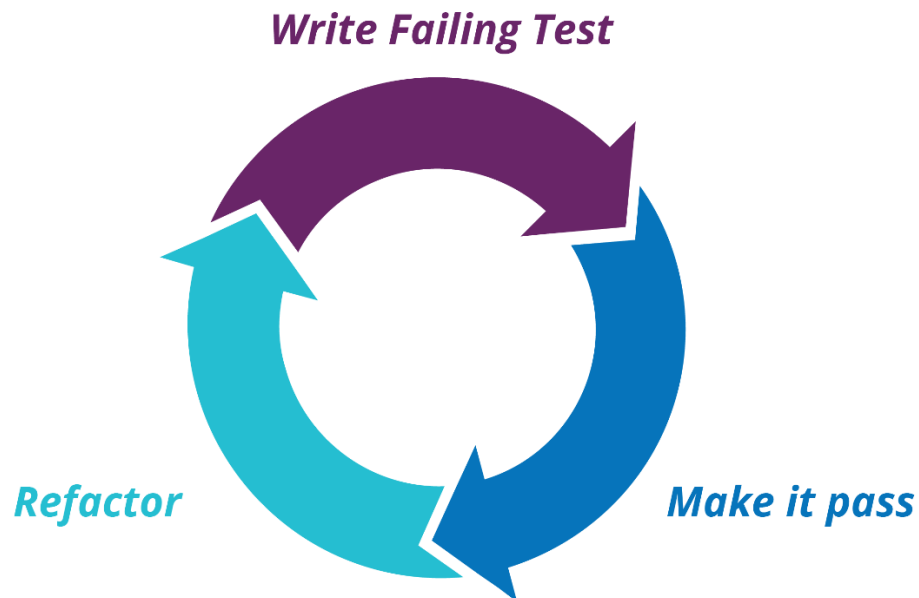
The link for code and test: <https://github.com/melkong194/Hangman>

2. Context and Background

2.1 Introduction of Testing Driven Development

Testing Driven Development (TDD) is a software development process that relies on the repetition of a very short development cycle:

- Firstly, writing a test case to test the requirement of the function.
- Secondly, produce the code to pass the test.
- Finally, refactoring the code.



This process of TDD is to test each logic of the program so TDD ensure the program meets requirements, preventing the cost for big bug and the possible solution is the minimum amount of code to pass the test. This help to avoid duplicating in writing code, clearer and better design of program. TDD becomes the good practice for developers because it helps developers get accomplishment feelings when each test pass or show them problems when testing results shows improper execution of functions.

Not similarity to traditional testing, TDD is 100% coverage test. It can be applied to test every logic in our program.

2.2 Hangman Game

How to play Hangman Game:

One word will be generated randomly. The word is hidden and only shown by a number of '*' to present of each letters of the words. Based on the length of the word, user will guess the word by entering a letter in turn. Initially, user has 5 lives and their lives will be decreased by one when making a wrong guess. Once user makes correct guess, the letter of that guess in all position will be revealed. If user makes guess correctly all letters in the words, they will win the game. However, when they got 5 times of incorrect guess, their lives will be 0 and lose the game.

Steps to create Hangman program: some steps are broken by the rules of the game.

- Pick a word randomly.
- Allow user to input
- Validate the input
- Compare the input to the word
- Check win or not.

2.3 Report Introduction

This report is to present the process of applying TDD to develop the hangman program. The Junit test cases will be specified the behaviors of functions in java code of hangman program.

3. Design

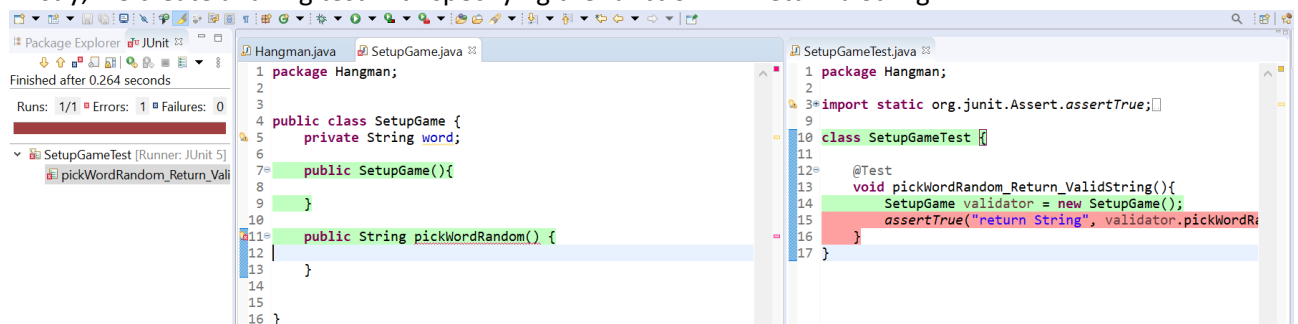
3.1 Requirements

Step	Functions	Requirements
1	Pick a word randomly	Pick a word in a list of words randomly and return a String.
2	Allow user to input	Allow user inputs a String
3	Validate the input	If the input is an alphabet then returns true, else returns false.
4	Compare the input to the word	If input is in words and not guessed before then returns true, else return false.
5	Check win or not	If all letters in the word are guessed then return true, else return false.

3.2 Design and approach

FUNCTION 1: Pick a word randomly

Firstly, we create a failing test with specifying the function will return a String.



Then we produce the code to pass the test.

```

1 package Hangman;
2
3
4 public class SetupGame {
5     private String word;
6
7     public SetupGame(){}
8
9
10
11     public String pickWordRandom() {
12         int pos=(int)(Math.random()*WordList.values().length)
13         return WordList.values()[pos].toString();
14     }
15
16
17 }

```

```

1 package Hangman;
2
3 import static org.junit.Assert.assertTrue;
4
5
6
7
8
9
10 class SetupGameTest {
11
12     @Test
13     void pickWordRandom_Return_ValidString(){
14         SetupGame validator = new SetupGame();
15         assertTrue("return String", validator.pickWordRandom() != null);
16     }
17 }

```

Package Explorer: JUnit

Finished after 0.27 seconds

Runs: 1/1 Errors: 0 Failures: 0

SetupGameTest [Runner: JUnit 5]

FUNCTION 2: Allow user to input

Function 2 is similar to the function 1, we create a failing test with specifying the function will return a String.

```

1 package Hangman;
2
3
4 public class SetupGame {
5     private String word;
6
7     public SetupGame(){
8         word = pickWordRandom();
9         int letterNumber = word.length();
10        for(int i=0; i<letterNumber;i++)
11            System.out.print("*");
12    }
13
14
15
16     public String pickWordRandom() {
17         int pos=(int)(Math.random()*WordList.values().length)
18         return WordList.values()[pos].toString();
19     }
20
21     public String inputLetter(){
22         System.out.println("\nGuess a letter in the word:");
23     }
24 }

```

```

1 package Hangman;
2
3 import static org.junit.Assert.assertTrue;
4
5
6
7
8
9
10 class SetupGameTest {
11
12     @Test
13     void pickWordRandom_Return_ValidString(){
14         SetupGame validator = new SetupGame();
15         assertTrue("return String", validator.pickWordRandom() != null);
16     }
17
18     @Test
19     void inputLetter_ReturnsValidString(){
20         SetupGame validator = new SetupGame();
21         assertTrue("return String", validator.inputLetter() != null);
22     }
23 }

```

Package Explorer: JUnit

Finished after 0.282 seconds

Runs: 2/2 Errors: 1 Failures: 0

SetupGameTest [Runner: JUnit 5] (pickWordRandom_Return_ValidString)

Then we produce the code to pass the test.

```

1 package Hangman;
2
3 import java.io.BufferedReader;
4 import java.io.InputStreamReader;
5
6 public class SetupGame {
7     private String word;
8
9     public SetupGame(){
10        word = pickWordRandom();
11        int letterNumber = word.length();
12        for(int i=0; i<letterNumber;i++)
13            System.out.print("*");
14    }
15
16
17     public String pickWordRandom() {
18         int pos=(int)(Math.random()*WordList.values().length)
19         return WordList.values()[pos].toString();
20     }
21
22     public String inputLetter() throws Exception {
23         System.out.println("\nGuess a letter in the word:");
24         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
25         return br.readLine();
26     }
27 }

```

```

1 package Hangman;
2
3 import java.io.BufferedReader;
4 import java.io.InputStreamReader;
5
6
7
8
9
10 import java.util.regex.Matcher;
11
12 import org.junit.jupiter.api.Test;
13
14 import junit.framework.Assert;
15
16 class SetupGameTest {
17
18     @Test
19     void pickWordRandom_Return_ValidString(){
20         SetupGame validator = new SetupGame();
21         assertTrue("return String", validator.pickWordRandom() != null);
22     }
23
24     @Test
25     void inputLetter_ReturnsValidString() throws Exception {
26         SetupGame validator = new SetupGame();
27         String input = "A";
28         ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes());
29         System.setIn(in);
30         assertTrue("return String", validator.inputLetter() != null);
31     }
32 }

```

Package Explorer: JUnit

Finished after 0.307 seconds

Runs: 2/2 Errors: 0 Failures: 0

SetupGameTest [Runner: JUnit 5] (0.069 s)

FUNCTION 3: Validate the input

The conditions in the function 3 requires more test cases to specify the requirements.

Apply the TDD, we also produce the failing test case firstly to ensure no empty input.

The screenshot shows an IDE with three panels. The left panel shows the Package Explorer with a test run summary: 'Finished after 0.212 seconds', 'Runs: 3/3', 'Errors: 0', 'Failures: 1'. The middle panel shows the source code for `Hangman.java` and `SetupGame.java`. The `checkInput` method in `SetupGame.java` is highlighted, showing it returns `true` for any non-empty string. The right panel shows the test code in `SetupGameTest.java`, where the `checkInput_notNull()` test is failing. The failure trace indicates an `AssertionFailedError` because the expected value was `<false>` but the actual value was `true`.

```

6 public class SetupGame {
7     private String word;
8
9     public SetupGame(){
10         word = pickWordRandom();
11         int letterNumber = word.length();
12         for(int i=0; i<letterNumber;i++)
13             System.out.print("*");
14     }
15
16     public String pickWordRandom() {
17         int pos=(int)(Math.random()*WordList.values().length);
18         return WordList.values()[pos].toString();
19     }
20
21     public String inputLetter() throws Exception {
22         System.out.println("\nGuess a letter in the word");
23         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
24         return br.readLine();
25     }
26
27     public Boolean checkInput(String s) {
28         return true;
29     }
30 }

```

```

16 class SetupGameTest {
17
18     @Test
19     void pickWordRandom_Return_ValidString(){
20         SetupGame validator = new SetupGame();
21         assertTrue("return String", validator.pickWordRandom().length() > 0);
22     }
23
24     @Test
25     void inputLetter_ReturnsValidString() throws Exception {
26         SetupGame validator = new SetupGame();
27         String input = "A";
28         InputStream in = new ByteArrayInputStream(input.getBytes());
29         System.setIn(in);
30         assertTrue("return String", validator.inputLetter().length() > 0);
31     }
32
33     @Test
34     void checkInput_notNull() throws Exception {
35         SetupGame validator = new SetupGame();
36         String example = "";
37         assertEquals(false, validator.checkInput(example));
38     }
39
40 }
41

```

Then, minimum code to pass the test.

The screenshot shows the same IDE after modifying the code. The left panel shows the test run summary: 'Finished after 0.299 seconds', 'Runs: 4/4', 'Errors: 0', 'Failures: 0'. The middle panel shows the updated `checkInput` method in `SetupGame.java`, which now returns `false` for an empty string. The right panel shows the updated test code in `SetupGameTest.java`, where the `checkInput_notNull()` test is now passing, and a new test `checkInput_1Letter_first()` has been added.

```

6 public class SetupGame {
7     private String word;
8
9     public SetupGame(){
10         word = pickWordRandom();
11         int letterNumber = word.length();
12         for(int i=0; i<letterNumber;i++)
13             System.out.print("*");
14     }
15
16     public String pickWordRandom() {
17         int pos=(int)(Math.random()*WordList.values().length);
18         return WordList.values()[pos].toString();
19     }
20
21     public String inputLetter() throws Exception {
22         System.out.println("\nGuess a letter in the word");
23         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
24         return br.readLine();
25     }
26
27     public Boolean checkInput(String s) {
28         if(s == "") return false;
29         return true;
30     }
31 }
32
33

```

```

21     assertTrue("return String", validator.pickWordRandom().length() > 0);
22 }
23
24     @Test
25     void inputLetter_ReturnsValidString() throws Exception {
26         SetupGame validator = new SetupGame();
27         String input = "A";
28         InputStream in = new ByteArrayInputStream(input.getBytes());
29         System.setIn(in);
30         assertTrue("return String", validator.inputLetter().length() > 0);
31     }
32
33     @Test
34     void checkInput_notNull() {
35         SetupGame validator = new SetupGame();
36         String example = "";
37         assertEquals(false, validator.checkInput(example));
38     }
39
40     @Test
41     void checkInput_1Letter_first() {
42         SetupGame validator = new SetupGame();
43         String example = "a";
44         assertTrue(validator.checkInput(example));
45     }
46
47 }
48

```


Testing for a single letter input, it works properly and pass the test.

The screenshot shows an IDE with three panels. The left panel displays the Package Explorer and a JUnit runner summary: 'Finished after 0.626 seconds', 'Runs: 3/3', 'Errors: 0', 'Failures: 0'. The middle panel shows the source code for `Hangman.java` and `SetupGame.java`. The right panel shows the test code in `SetupGameTest.java`. The tests listed in the runner are:

- `SetupGameTest [Runner: JUnit 5] (0.325 s)`
- `pickWordRandom_Return_ValidString() (0.088 s)`
- `inputLetter_ReturnsValidString() (0.053 s)`
- `checkInput_notNull() (0.184 s)`

By repeating the process of TDD, test cases are written before the produce of code, the function is tested for other requirements, such as, testing for more than 1 letter inputs, special letters, digits, leading space before inputs. When we produce code to pass all test cases, we have a complete function.

Before producing code:

The screenshot shows the same IDE setup but with a different state. The JUnit runner summary shows 'Runs: 6/6', 'Errors: 0', 'Failures: 1'. The test results list includes:

- `SetupGameTest [Runner: JUnit 5] (0.148 s)`
- `pickWordRandom_Return_ValidString() (0.053 s)`
- `inputLetter_ReturnsValidString() (0.003 s)`
- `checkInput_1Letter_first() (0.003 s)`
- `checkInput_1Letter_third() (0.023 s)`
- `checkInput_notNull() (0.030 s)`
- `checkInput_1Letter_second() (0.031 s)`

The Failure Trace indicates a `java.lang.AssertionError` at `Hangman.SetupGameTest.checkInput_1Letter_third(S`. The source code in the middle panel shows the `checkInput` method, and the test code in the right panel shows the `checkInput_1Letter_third` test case, which is highlighted in red, indicating it failed.

After producing code to pass all tests:

The screenshot shows the Eclipse IDE with the Package Explorer on the left, displaying the test results for the SetupGameTest class. The main editor shows the code for Hangman.java and SetupGameTest.java. The test results show that all tests passed successfully.

```

Package Explorer
Finished after 0.415 seconds
Runs: 10/10 Errors: 0 Failures: 0

SetupGameTest [Runner: JUnit 5] (0.144 s)
  pickWordRandom_Return_ValidString() (0.068 s)
  inputLetter_ReturnsValidString() (0.003 s)
  checkInput_isAlphabet_1st() (0.003 s)
  checkInput_isAlphabet_2nd() (0.003 s)
  checkInput_isAlphabet_3rd() (0.002 s)
  checkInput_1Letter_4th() (0.002 s)
  checkInput_1Letter_first() (0.002 s)
  checkInput_1Letter_third() (0.003 s)
  checkInput_notNull() (0.036 s)
  checkInput_1Letter_second() (0.021 s)

Failure Trace

Hangman.java
12  import java.io.*;
13  import java.util.*;
14  System.out.println("*****");
15  }
16
17  public String pickWordRandom() {
18      int pos = (int) (Math.random() * WordList.values().length);
19      return WordList.values()[pos].toString();
20  }
21
22  public String inputLetter() throws Exception {
23      System.out.println("\nGuess a letter in the word:");
24      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
25      return br.readLine();
26  }
27
28  public Boolean checkInput(String s) {
29      if (s == null)
30          return false;
31      else
32          s = s.trim();
33
34      if (s.length() > 1)
35          return false;
36
37      return s.matches("[a-zA-Z]");
38  }
39  }

SetupGameTest.java
62  void checkInput_1Letter_4th() {
63      SetupGame validator = new SetupGame();
64      String example = "B";
65      assertTrue(validator.checkInput(example));
66  }
67
68  @Test
69  void checkInput_isAlphabet_1st() {
70      SetupGame validator = new SetupGame();
71      String example = "1";
72      assertFalse(validator.checkInput(example));
73  }
74
75  @Test
76  void checkInput_isAlphabet_2nd() {
77      SetupGame validator = new SetupGame();
78      String example = " ";
79      assertFalse(validator.checkInput(example));
80  }
81
82  @Test
83  void checkInput_isAlphabet_3rd() {
84      SetupGame validator = new SetupGame();
85      String example = "F";
86      assertTrue(validator.checkInput(example));
87  }
88  }
89  }

```

FUNCTION 4 and 5:

The process for function 4 and 5 is similar to 1,2,3.

The screenshot shows the Eclipse IDE with the Package Explorer on the left, displaying the test results for the SetupGameTest class. The main editor shows the code for Hangman.java and SetupGameTest.java. The console output shows the execution of the tests.

```

JavaCode - Hangman/src/Hangman/SetupGame.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer
Finished after 0.594 seconds
Runs: 12/12 Errors: 0 Failures: 0

SetupGameTest [Runner: JUnit 5] (0.252 s)

Hangman.java
5  import java.io.*;
6
7  public class SetupGame {
8      private String word;
9      private String hiddenWord;
10     private int lives = 5;
11
12     public SetupGame() {
13         word = "monday";
14         hiddenWord = new String(new char[word.length()]);
15     }
16
17     public Boolean compareGuess(String guess) {
18         guess = guess.toLowerCase();
19         Boolean correctGuess = false;
20         for (int i = 0; i < word.length(); i++) {
21             if (word.charAt(i) == guess.charAt(0)) {
22                 hiddenWord = hiddenWord.substring(0, i) + guess.charAt(0) + hiddenWord.substring(i+1, word.length());
23                 correctGuess = true;
24             }
25         }
26         System.out.println(hiddenWord);
27         return correctGuess;
28     }
29
30     public String pickWordRandom() {
31         // pick a word from WordList
32         int pos = (int) (Math.random() * WordList.values().length);
33         return WordList.values()[pos].toString();
34     }
35 }

SetupGameTest.java
51
52
53  @Test
54  void checkInput_isAlphabet_2nd() {
55      assertTrue(validator.checkInput(" "));
56  }
57
58  @Test
59  void checkInput_isAlphabet_3rd() {
60      assertTrue(validator.checkInput("F"));
61  }
62
63  @Test
64  void compareGuess_1st() {
65      assertTrue(validator.compareGuess("c"));
66  }
67
68  @Test
69  void compareGuess_2nd() {
70      assertTrue(validator.compareGuess("n"));
71  }
72
73  @Test
74  void compareGuess_3rd() {
75      assertTrue(validator.compareGuess("A"));
76  }
77  }
78  }

Problems Declaration Console Coverage
<terminated> SetupGameTest [JUnit] C:\Program Files\Java\jre1.8.0_251\bin\javaw.exe (29 Aug 2020, 8:16:20 pm - 8:16:23 pm)
*****
*****
*****
Guess a letter in the word:

```

REFACTORING THE CODE:**Refactoring the test cases:****@Before**

```

147 SetupGameTest.java
40 @Test
41 void checkInput_1Letter_first(){
42     SetupGame validator = new SetupGame();
43     String example = "a";
44     assertTrue(validator.checkInput(example));
45 }
46
47 @Test
48 void checkInput_1Letter_second(){
49     SetupGame validator = new SetupGame();
50     String example = "ab";
51     assertFalse(validator.checkInput(example));
52 }
53
54 @Test
55 void checkInput_1Letter_third(){
56     SetupGame validator = new SetupGame();
57     String example = " B ";
58     assertTrue(validator.checkInput(example));
59 }
60
61 @Test
62 void checkInput_1Letter_4th(){
63     SetupGame validator = new SetupGame();
64     String example = " B ";
65     assertTrue(validator.checkInput(example));
66 }
67

```

@After

```

15
16 class SetupGameTest {
17     SetupGame validator = new SetupGame();
18
19     @Test
20     void pickWordRandom_Return_ValidString(){
21         assertTrue("return String", validator.pickWordRandom_Return_ValidString());
22     }
23
24     @Test
25     void inputLetter_ReturnsValidString() throws Exception {
26         String input = "A";
27         InputStream in = new ByteArrayInputStream(input.getBytes());
28         System.setIn(in);
29         assertTrue("return String", validator.inputLetter_ReturnsValidString());
30     }
31
32     @Test
33     void checkInput_notNull(){
34         String example = "";
35         assertEquals(false, validator.checkInput(example));
36     }
37
38     @Test
39     void checkInput_1Letter_first(){
40         String example = "a";
41         assertTrue(validator.checkInput(example));
42     }
43

```

Refactoring the Programming Code:

Because after user got win or lose, the game will be repeated. Hence, we refactor the code and extract it into new method to reuse it and get better design for program.

@Before refactoring

The screenshot shows the IDE with `Hangman.java` open. A code block from line 17 to 32 is selected, containing a `do` loop for game logic. A context menu is open over the selection, with `Refactor` highlighted. A secondary menu is also open, showing `Extract Method...` as the selected option. The background code includes `SetupGame` constructor and `compareGuess` method.

@After Refactoring.

```
Hangman.java SetupGame.java
6
7 public class SetupGame {
8     private String word;
9     private String hiddenWord;
10    private int lives;
11
12    public SetupGame() {
13        while (true) {
14            System.out.println("++++NEW GAME++++");
15            word = pickWordRandom().toLowerCase();
16            lives = 5;
17            hiddenWord = new String(new char[word.length()]).replace("\0", "*");
18            play();
19        }
20    }
21
22    public void play() {
23        do {
24            System.out.println("=== LIVES: " + lives + " ===");
25            System.out.println("The word: " + hiddenWord);
26
27            // Validate input
28            String guess = inputLetter();
29            while (!checkInput(guess)) {
30                System.out.println("Invalid input! Please make guess again!!");
31                guess = inputLetter();
32            }
33
34            // Result for each letter input.
35            if (compareGuess(guess)) {
36                System.out.println("Good Guess!!\n");
37            } else {
```

4. Reference

https://en.wikipedia.org/wiki/Test-driven_development