HANGMAN

Berk Can BALATACI
Linnaeus University (Erasmus Exchange Student)
18.04.2019

Contents

1 | Revision History

Date	Version	Description	Author
08.02.2019	1.0	Project plan and Skeleton codes are published.	Berk Can BALATACI
22.02.2019	1.1	UML diagrams, time log and codes of playable version of Hangman are published.	Berk Can BALATACI
08.03.2019	1.2	Manual test-cases, unit test-case classes and results are published.	Berk Can BALATACI
16.04.2019	1.3	Several parts are updated: • General Information • Project Plan • Risk Analysis • Time Log	Berk Can BALATACI
16.04.2019	1.4	Problems from previous iterations are handled. Project is finished.	Berk Can BALATACI

2 | General Information

Project Summary	
Project Name	Project ID
Hangman	bb222ji_1DV600
Project Manager	Main Client
Berk Can BALATACI	English Vocabulary Enthusiasts
Key Stakeholders	
Project Manager Software Architect Software Developer End users	
Executive Summary	
The Hangman Game project is a terminal based application that can run on any personal computer. The game comprises of guessing letters of a hidden word where each correct guess reveals the letter in the word. An incorrect guess adds a part to man being hanged. The game ends if all parts of the man hanged or all letters are guessed. The project is being developed in order to make people fun and discover & increase their English vocabulary knowledge at the same time.	

3 | Vision

The Hangman Game is a text based fun game such that people of all ages want to play it. Players can race between other players who are played on the same computer and also they can learn new words and definitions.

When the program runs, the player will be welcomed with a menu that have choices like start the game, add a new word to game's dictionary. If he/she selects the start choice than name of the player will be asked and then game will start. At the beginning of the game, a word from a predefined dictionary is randomly be picked and the underline signs is displayed as the number of letters of the word. Then player tries to guess the word by suggesting letter after letter. Players have fixed wrong prediction limit. After a round is finished, system can display the definition of the word if user wants to see.

Reflections: *First time for me to write a vision of a project. So I am not sure that if I mention what is the application about or how can I give a motivation to stakeholders of the project. At first paragraph, I give few motivation information about game that include important features may differ from other competitive games. At second paragraph, I mention what is the program about and how will it look after the project is finished.*

4 | Project Plan

4.1 Introduction

The project is about creating a well-known Hangman game with plan-driven approach. The application will be a terminal based application that can run on any personal computer.

4.2 Justification

This game will entertain people while they are learning new words and their definitions. The game's dictionary is huge (54467 words with definitions) so that it is almost impossible for a gamer to see same words that he/she already encountered. In this way, the game is more challenging and educational.

4.3 Stakeholders

Project Manager -- The person responsible for the project team. Project Manager wants to see that the project plan is followed by team and the application will be delivered before deadline.

Software Architect -- The person responsible for analyzing, design and modelling. This person will design UML Diagrams to model Project features.

Software Developer -- The person responsible for implementation and testing. Developer will implement all required features. This person wants to write well structured code for better understandability, changeability, testability.

End user -- The person knows how to play terminal based Hangman game. End user can suggest which features are fun or useful and after end of development this person can make comments about the game.

4 | Project Plan

4.4 Resources

- A computer which has Intel i7 Processor, 8GB Ram and Windows 10 Operating System is used to create the application.
- The program is coded in Java language with Eclipse IDE.
- GitHub is used for version control and storing documents.
- Draw.io is used for drawing diagrams.
- Project will take 50 days to end.
- 1 employee works on this project. In this way, project manager, software architect and software developer will be the same person.
- It will cost 0 SEK as the employee works on this project voluntarily for educational purposes.

4.5 Hardware and Software Requirements

Hardware

- A working computer that can run Java Virtual Machine.
- Mouse and keyboard.

Software

- An operating system which Java 8 or higher is installed.

4 | Project Plan

4.6 Overall Project Schedule

Deadlines

- Start --- 01.02.2019
- Planning Phase End --- 08.02.2019 (Iteration 1)
- Design Phase End --- 21.02.2019 (Iteration 2)
- Implement & Test Phase End --- 08.03.2019 (Iteration 3)
- Complete Project --- 18.04.2019 (Iteration 4)

4.7 Scope, Constraints and Assumptions

Scope: The project is only in English language. The application keeps definition of words in a csv file. There is no exact word type, it can be a noun, verb, adj, etc. The main scenario of the game is a text based well-known Hangman Game. There are 2 features in game: displaying the definition of the word after a round finished, adding a new word to game's dictionary. The application is implemented and tested in Windows.

Out of Scope: The application does not have a graphical user interface as it is a terminal based application. This is because developer lacks of knowledge about it. Because of the fact that this application only tried with Windows, it may cause a problem in other Operating Systems.

Constraints: Time was limited. There is only 1 employee works on this project. Developer lacks of knowledge to code a Graphical User Interface in Java. The game runs in a terminal environment. The application must be fast and easy to play.

Assumption: Users must have Java8 or later and they must read installing instructions on GitHub readme file.

4 | Project Plan

Reflections: *At first, some of the parts of Project Plan that I have to write about was not clear for me. After a couple of searches, I think I understand most of the parts but I am not sure if I should write more information about Scope, Constraints and Assumptions of the Project as I do not want to write about same things mentioned above parts. As I don't know how can I calculate what is minimum system requirements to run this game, I could not write exact amount of ram, gpu, etc.*

5 | Iterations

This project consists of 4 iterations as below.

5.1 Iteration 1

In this iteration Project Plan was built. Several General Information about project was reported. Vision of the project was determined. A Project Plan and Iteration Analysis were created. Some skeleton code was implemented. A GitHub repo was opened and both Project Plan and Skeleton code are released.

- Write Vision 1:00
- Write Project Plan 3:00
- Analyze Iterations 4:00
- List Risks & Strategies to Avoid them 1:00
- Start coding (Skeleton code) 2:00
- Create a repo and upload project plan 0:30

5.2 Iteration2

In this iteration, Project features are modelled using Unified Modelling Language (UML). The result diagrams of this iteration are be added to the project documentation. Thanks to these diagrams, the application are implemented in the way modelled. End of this iteration, there was a playable version of the game in a basic style.

- Define and make time plan for tasks 0:15
- Learn and draw Use Case Diagram 1:15
- Read Larman guidelines about fully dressed use cases and write Fully Dressed UC for PlayGameUC 1:45

- Draw State Machine Diagram for PlayGame 1:30
- Implement Basic Game 4:30
- Create Class Diagram 0:30

5.3 Iteration 3

In this iteration use-cases are tested by writing and running dynamic manual test-cases. Test-cases for UseCase1(Start Game) and UseCase2(Play Game) are created. Moreover, automated unit tests are wrote and run for all methods which are implemented so far.

- Make time plan 0:30
- Writing and Running Manual Test Cases 3:00
- Writing and Running Unit Tests 4:00
- Code inspection 0:45
- Test Report 0:30
- Write reflection 0:30

5.4 Iteration 4

In this iteration, the complete project will be the outcome. First the problems from previous iterations will be handled. Then, the steps in iteration 1,2 and 3 will be reiterated for new features and project will be considered as a whole.

Make plan for iteration 4

- Think about tasks and write a detailed plan 0:45

5 | Iterations

Handle problems on iteration 1,2

- Change main client and executive summary 0:30
- Update Project Plan 2:00
- Update Iterations Analysis 0:45
- Update Risks & Strategies 1:00
- Change time logs' structure and add comparison 0:30
- Fix fully dressed use case 0:15

Implement the feature: displaying definition of the word

- Write fully dressed use case for this feature 0:30
- Write code for this feature 1:15
- Write and run manual test cases for this feature 0:45

Implement the feature: adding a new word to dictionary

- Write fully dressed use case for this feature 0:30
- Write code for this feature 1:00
- Write automated unit tests for this feature 1:00

Finish project

- Update general test report of the project 0:15
- Update use case diagram 0:30
- Update class diagram 0:30
- Make code structure more readable 0:15
- Check document structure 0:15
- Make code executable 0:45
- Upload Documents to GitHub 0:15

6 | Risk Analysis

6.1 List of risks

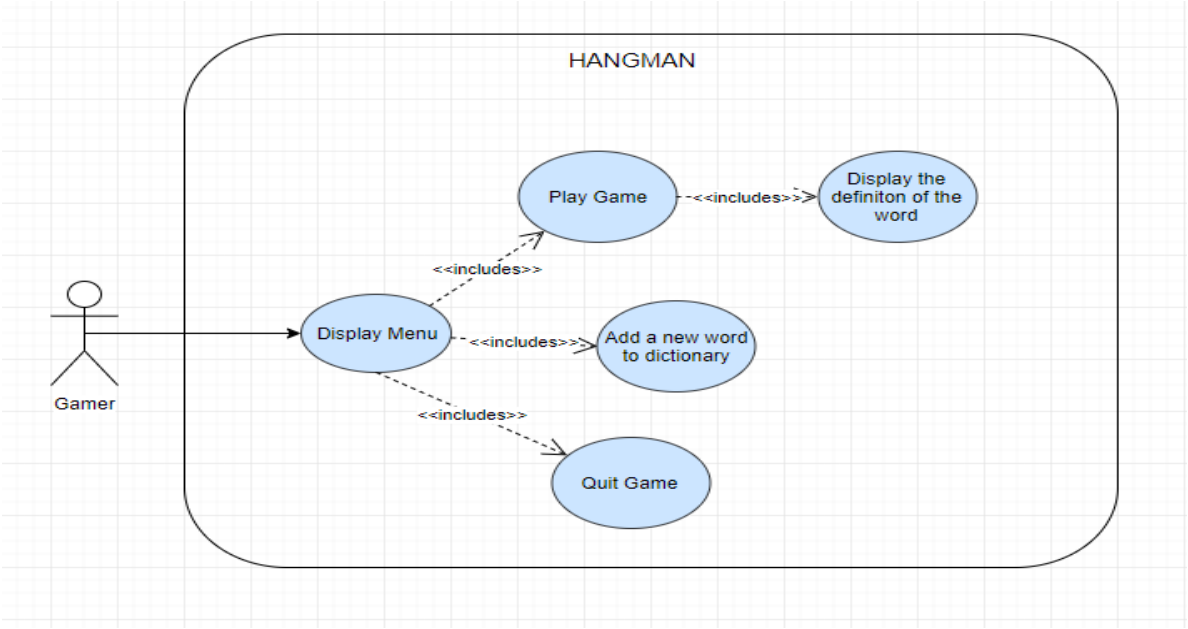
1) Developer may not have enough skills required for implementation.
   (Probability: Low) (Impact: Serious)
2) Project does not meet its requirements.
   (Probability: Medium) (Impact: Serious)
3) The time required to develop the application can be underestimated. This may project can't be completed before the deadline.
   (Probability: Medium) (Impact: Catastrophic)
4) Hardware crashes.
   (Probability: Low) (Impact: Catastrophic)

6.2 Strategies

1) Developer must plan time to train required skills. Features which are impossible to implement should be removed.
2) Specified requirements should be understood completely. Before project completed, each iteration must be re-iterated.
3) Employees may work more hours than estimated if iterations is behind schedule and if deadlines are passed then there is no other solution except asking client for more time.
4) Project files should be uploaded to GitHub frequently and hardware should be maintained regularly. If the hardware break then it should be repaired as soon as possible as there is 1 computer as resource.

Reflections: *I think Risk Analysis is a bit hard for me to do for this project. Because this was first time for me to document risks about a project. At first, I wrote some functionalities in the game like "A person can play by typing another player's name. " that I thought they are project risks but then I learned they are not project risks so I removed them.*

Use Case Diagram for Hangman



Use Case UC1: Start Game

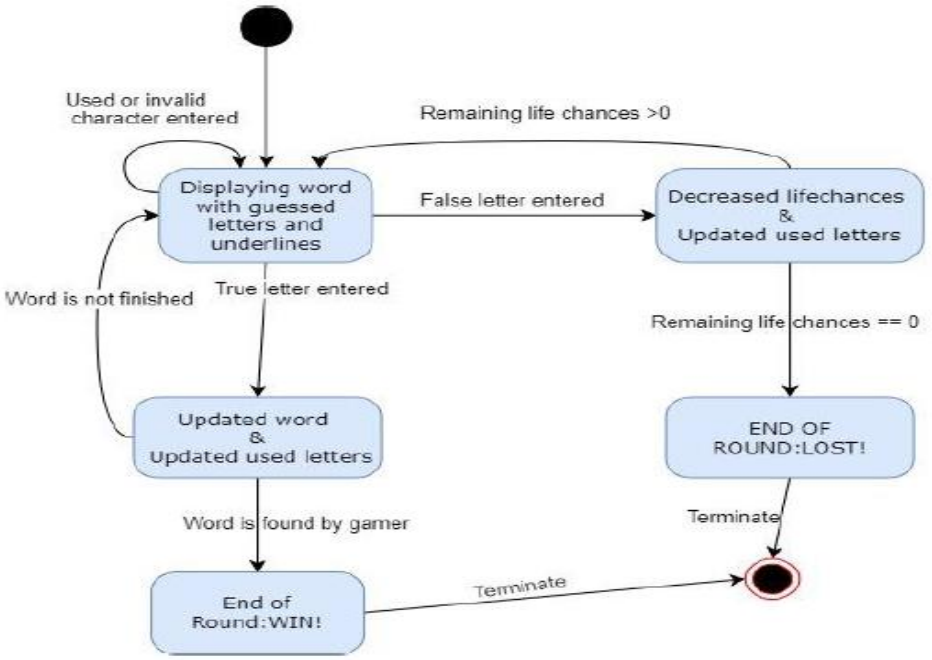
Level: User goal
Primary Actor: Gamer
Stakeholders and Interests:
– Gamer: Wants to start an error-free game.
Preconditions: None.
Postconditions: The game menu is shown.
Main Scenario:

1. Starts when user runs the application.
2. The system shows the game menu to gamer and asks for choice.
3. Gamer selects to start a round.

Alternative Scenarios:

3.1 Gamer selects to add a new word to dictionary.
1.Go to UC 4.
3.2 Gamer selects to quit the game.
1.The system terminates the game.

State Machine Diagram for "Play Game"



Use Case UC2: Play Game

Level: User goal
Primary Actor: Gamer
Stakeholders and Interests:
– Gamer: Wants to play an error-free game.
Preconditions: Gamer selects the Play Game choice on the menu.
Postconditions: Gamer plays the game. Results of the game is shown.
Main Scenario:

1. Starts when gamer selects to play a game round.
2. The system picks a random word and displays the number of letters of the word by underlines.
3. The system wait user to type a lowercase letter.
4. The gamer guesses a correct letter.
5. The system displays the word with correct guessed letters and underlines.

6. The gamer finds the correct word by guessing every letter and at least 1 life chance.
7. The system displays the points gamer gained depending on remaining life chances.
8. The gamer selects to see the definition of the word.
9. The system shows the definition of the word.
10. The system displays the menu of the game.

Alternative Scenarios:

4.1 The gamer guesses a wrong letter.
1. The system decreases a life chance of the gamer.
2. Go to 3

4.2 The gamer types an unexpected character.
1. The system presents an error message.
2. Go to 3

6.1 The gamer wastes his all life chances.
1. The system displays the correct word.
2. Go to 8

6.2 Games continues as the gamer still has life chances.
1. Go to 3

8.1 The gamer selects to not see the definition of the word.
1. Go to 10

Use Case UC3: Display Definition of Word

Level: User goal

Primary Actor: Gamer

Stakeholders and Interests:

– Gamer: Wants to see definition of the word.

Preconditions: Gamer finishes a round.

Postconditions: Definition of the word is shown.

Main Scenario:

1. Gamer finishes a round.
2. The system asks gamer to see the definition of the word.
3. The gamer selects “Yes”.
4. The system displays the definition of the word.
5. The system shows the menu.

Alternative Scenarios:

3.1 The gamer selects “No”.

1. Go to 5.

Use Case UC4: Add a new word to dictionary

Level: User goal

Primary Actor: Gamer

Stakeholders and Interests:

– Gamer: Wants to add a new word to dictionary.

Preconditions: Gamer selects add a new word to dictionary.

Postconditions: The word is added to dictionary.

Main Scenario:

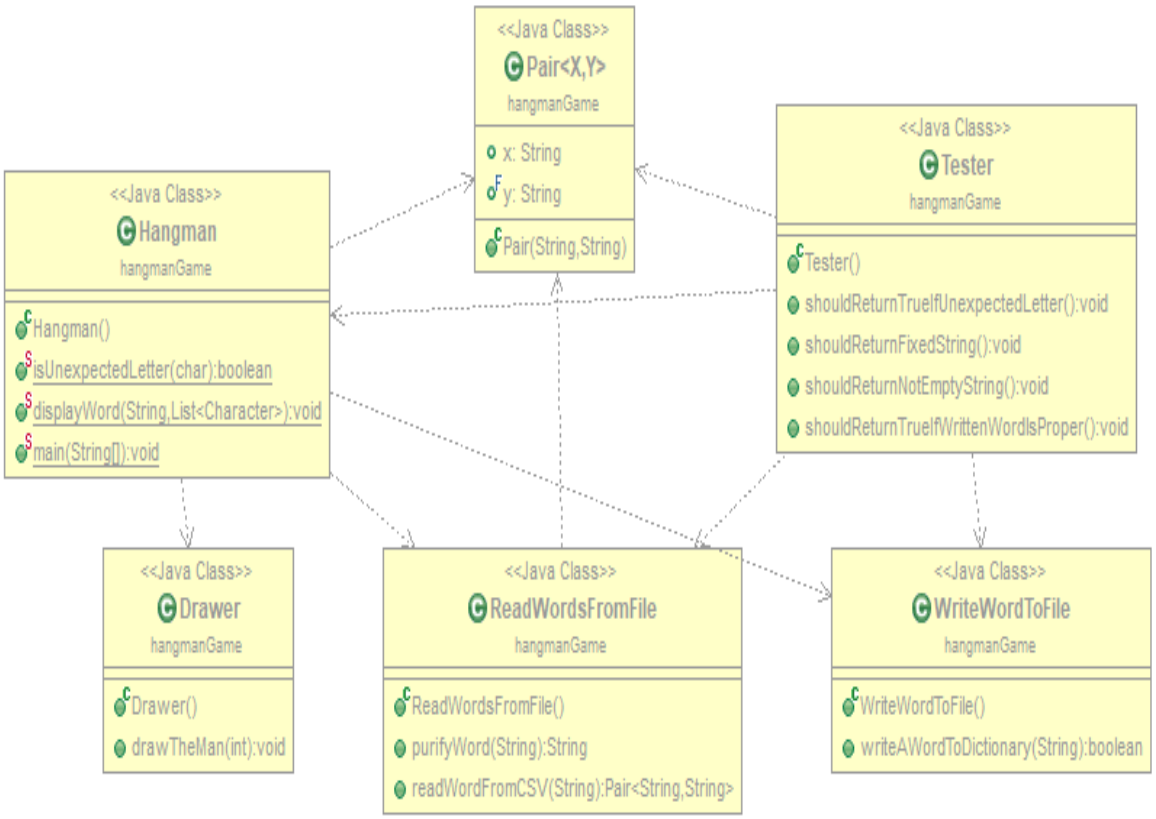
1. Gamer selects add a new word to dictionary from menu.
2. The system asks gamer to type the word.
3. The system asks gamer to definition of this word.
4. The system writes the word with its definition to the bottom row of dictionary.

Alternative Scenarios:

3.1 The gamer types an incorrect word.

1. Go to UC1.

Class Diagram of Final Implementation



Test Plan

Objective

The objective of the tests is check the program if it meets its requirements.

What to test and how?

The aim is to test use-cases by writing and running dynamic manual test-cases. I write test-cases for UC1(Start Game), UC2(Play Game) and UC3(Display Definition) as they are main functionalities of the program. Moreover, I write and run automated unit tests for all methods.

Manual Test-Cases

Test-Case 1.1: Start a round

Use case: UC1 Start Game
Scenario: Start a round successfully
Description: The first step of UC1 is tested where a gamer starts a round.
Precondition: “dictionary.csv” file must be in project folder.

Test Steps

- Run the application
- The system shows the main menu and expects an input.
- Type ‘1’ and press enter.

Expected

- The system picks a random word from “dictionary.csv” and starts the round.

Test Result: Test is passed successfully.

Test-Case 2.2: Win game

Use case: UC2 Play Game
Scenario: Win a round
Description: Main Scenario of UC2 is tested where a gamer plays the round and wins it.
Precondition: Starting a round choice must be selected. (TC1.1 was completed)
Note: *The word guessed is selected as "bass" for testing.*

Test Steps

- The program shows " ***** The round is starting... *****

  Guess this word : _ _ _ _
  Guess a letter in the word :
- Type 'a' and press enter.
- System shows " _a _ _

  Guess a letter in the word : "
- Type 'b' and press enter.
- System shows " b a _ _

  Guess a letter in the word : "
- Type 's' and press enter.

Expected

- Score of gamer is displayed.
- Main menu is displayed.

Test Result: Test is passed successfully.

Test-Case 2.3 : Lose game

Use case: UC2 Play Game
Scenario: Lose a round
Description: Alternative Scenario of UC2 is tested where a gamer plays a round and loses it because of spending all 8 life chances.
Precondition: Starting a round choice must be selected. (TC1.1 was completed)
Note: *The word guessed is selected as "bass" for testing.*

Test Steps

- The program shows " ***** The round is starting... *****

  Guess this word : _ _ _ _
  Guess a letter in the word :
- Type 'b' and press enter.
- System shows " b _ _ _

  Guess a letter in the word : "
- Type 'e' and press enter.
- System shows " b _ _ _

  You have 7 life chances ! Guess a letter in the word : "
- Type 'r' and press enter.
- System shows " b _ _ _

  You have 6 life chances ! Guess a letter in the word : "
- Type 'r' and press enter.
- System shows " b _ _ _

  You have 5 life chances ! Guess a letter in the word : "
- Type 'k' and press enter.
- System shows " b _ _ _

  You have 4 life chances ! Guess a letter in the word : "
- Type 'c' and press enter.

- System shows “ b _ _ _

  You have 3 life chances ! Guess a letter in the word : ”
- Type ‘a’ and press enter.
- System shows “ b a _ _

  Guess a letter in the word : ”
- Type ‘n’ and press enter.
- System shows “ b a _ _

  You have 2 life chances ! Guess a letter in the word : ”
- Type ‘l’ and press enter.
- System shows “ b a _ _

  You have 1 life chances ! Guess a letter in the word : ”
- Type ‘t’ and press enter.

Expected

- Round is lost as gamer is out of life chances.
- The word gamer tried to find is displayed.
- Main menu is displayed.

Test Result: Test is passed successfully.

Test-Case 2.4 : Unexpected or already used letters cause re- prompt

Use case: UC2 Play Game
Scenario: Win a round
Description: An alternate scenario of UC2 is tested where a gamer tries to enter unexpected letters and is forced to enter a new letter.
Precondition: Starting a round choice must be selected. (TC1.1 was completed)

Test Steps

- The program shows “

  Guess this word : b _ _ _
  Guess a letter in the word : ”
- Type ‘b’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!

  Guess a letter in the word : ”
- Type ‘15’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!

  Guess a letter in the word : ”
- Type ‘C’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!

  Guess a letter in the word : ”
- Type ‘*’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!

  Guess a letter in the word :

Expected

- The system displays a warning for unexpected or used letters.

Test Result: Test is passed successfully.

Test-Case 3.1: Display definition of word

Use case: UC3 Display Definition of the Word
Scenario: Display the definition of the word successfully
Description: UC3 is tested where a gamer wants to see the definition of the word.
Precondition: Gamer finishes a round.

Test Steps

- The program shows " Type 1 if you want to see the definition of the word ? "
- Type '1' and press enter.

Expected

- The system shows the definition of the word.

Test Result: Test is passed successfully.

Test-Case 3.2: Do not display definition of word

Use case: UC3 Display Definition of the Word
Scenario: Do not display the definition of the word
Description: UC3 is tested where a gamer does not want to see the definition of the word.
Precondition: Gamer finishes a round.

Test Steps

- The program shows " Type 1 if you want to see the definition of the word ? "
- Type '2' and press enter.

Expected

- The system shows the menu.

Test Result: Test is passed successfully.

Test Report

Test traceability matrix and success

Test	UC1	UC2	UC3
Test-Case 1.1	1/OK	0	0
Test-Case 2.2	0	1/OK	0
Test-Case 2.3	0	1/OK	0
Test-Case 2.4	0	1/OK	0
Test-Case 3.1	0	0	1/OK
Test-Case 3.2	0	0	1/OK
COVERAGE & SUCCESS	1/OK	3/OK	2/OK

Reflection: I was curious about testing part because I always wanted to know how I should test a software in appropriate way. I did not know anything about testing environments like JUnit. With this iteration, I think I learned a bit why and how I should do it. In addition, while I was working on writing unit-tests I realized that I should use more methods instead of coding same thing with less methods since it is better for testing. Due to the fact that I did not know much about unit-testing, it took more time than I thought.

11 | Time log

Iteration 1

Job	Actual	Estimated
Write Vision	1:30	1:00
Write Project Plan	4:15	3:00
Analyze Iterations	2:45	4:00
List Risks & Strategies to Avoid them	0:45	1:00
Start coding (Skeleton code)	1:00	2:00
Create a repo and upload project plan with skeleton code	0:15	0:30

Comments: Writing vision and project plan took more time then I estimated. Because they both are important parts for this project documents and I spent too much time on brainstorming as I did not know what to mention as it was first time for me to do.

11 | Time log

Iteration 2

Job	Actual	Estimated
Define and make time plan for tasks	0:30	0:15
Learn and draw Use Case Diagram	1:45	1:15
Read Larman guidelines about fully dressed use cases and write Fully Dressed UC for PlayGameUC	2:30	1:45
Draw State Machine Diagram for PlayGame	1:30	1:30
Implement Basic Game	4:00	4:30
Create Class Diagram	0:30	0:30

Comments: I estimate less time for learning and writing fully dressed use cases. I think it is because of I did not think widely for making time plan.

11 | Time log

Iteration 3

Job	Actual	Estimated
Make test plan & time plan	1:00	0:30
Writing and Running Manual Test Cases	3:45	3:00
Writing and Running Unit Tests	5:30	4:00
Code inspection	0:15	0:45
Test Report	0:15	0:15
Write reflection	0:30	0:30

Comments: Due to the fact that I did not know much about unit-testing, it took more time than I thought. I spent more time on making test plan than I estimated as it was first time for me. Also, I spent less time than I estimated on code inspection as I did not code too much yet and while I was writing unit tests, I was kind of inspecting the code.

11 | Time log

Iteration 4

Job	Actual	Estimated
Think about tasks and write a detailed plan	1:00	0:45
Change main client and executive summary	0:30	0:30
Update Project Plan	2:15	2:00
Update Iterations Analysis	0:45	0:45
Update Risks & Strategies	1:15	1:00
Change time logs' structure and add comparison	0:45	0:30
Fix fully dressed use case	0:15	0:15
Write fully dressed use case for displaying definition	0:15	0:30
Write code for displaying definition	1:00	1:15
Write and run manual test cases for displaying definition	0:30	0:45
Write fully dressed use case for adding new word	0:30	0:30
Write code for adding new word	0:45	1:00
Write automated unit tests for adding new word	0:45	1:00
Update general test report of the project	0:15	0:15
Update use case diagram	0:15	0:30
Update class diagram	0:15	0:30
Make code look more steady	0:15	0:15
Fix document structure(font, size, spaces etc.)	0:30	0:15
Make code executable	1:15	0:45
Merge Documents and Upload to GitHub	0:30	0:15

Comments: In this iteration, I spent more time on writing a detailed plan than I did on other iterations. Thanks to experience from previous iterations and spending more time on planning, I got more satisfactory results between estimated and actual times except trying to make code executable.

12 | Handing in

GitHub url: https://github.com/berkbltc/bb222ji_1dv600/releases

Reflection and Comments for Iteration 4

In this iteration, I choose two new features to implement. The important one is displaying definition of word since this feature allows people to learn definitions of words and develop their vocabulary as the project promises from the beginning. The other feature is adding new word to dictionary, I choose this feature as it is easier to implement than other features I wanted. I was going to implement registration system and high score list but due to the lack of time I decided to exclude them as I want to focus design, test and documentation instead of coding.

I choose fully dressed use case for design both new features because fully dressed use cases are more detailed when you compare it to UML use case or state machine diagrams, so that developer can understand the feature clearly. Moreover, I wrote manual test cases for displaying definition as it prints the definition to screen so that I test it if the definition is correct or not by looking csv file manually. For the other feature, I decided to implement unit test as there are lots of different things that I want to send to function and it is easier to test that function with Junit framework. In the end, I think generally unit tests are more reliable and fast than manual tests as tester can get confused in long documents.

I try and spend more time then I estimated on making code executable but because of the fact that I use common-csv jars, and lacks of knowledge about it I could not do it.