SOFTWAREDEVELOPMENT HANGMAN

Anna Malmgren
Linnéuniversitetet, Software Technology

08/02/2019

Contents

1 | Revision History

Date	Version	Description	Author
08/02/2019	v1.0	Initial version.	Anna Malmgren

2 | General Information

Project Summary	
Project Name	Project ID
Hangman	am223rj _1DV600
Project Manager	Main Client
Anna Malmgren	Anna Malmgren
Key Stakeholders	
The end users/clients, project manager, main client, developers , organization owning the project.	
Executive Summary	
A console application of the game Hangman created for fun learning. Implemented over 4 iterations, using a project plan, risk analysis, UML modelling and testing.	

3 | Vision

Hangman is a console application built on the Node.js platform and developed for fun learning for anyone and everyone! Hangman creates a new exciting learning environment where only your fantasy sets the limits. Implementing different words to guess and trying to beat high scores makes broaden peoples vocabulary a piece of cake.

The basic idea of Hangman is that the player is going to guess a word by suggesting letter after letter. The player is presented with the number of letters in the word and a clue, but for every wrong guess the game is building a part of a man getting hanged. The number of wrongs that the player can have is depending on how many parts the hanged man consists of. If the man gets hanged, the player loses. Additionally, the player should only have a certain amount of time to guess a letter, and if he/she succeeds and wins the game there is also a possibility to reach a high score board.

Reflections:

The hardest part of writing the vison for me was trying make the vision motivating and inspiring for others. I am more a matter of fact person so what's inspiring to me may not be so inspiring to most others. I'm still not sure if this vision is motivating enough but I tried to find a balance between keeping the projects direction clear but still have some abstract parts so one can form new ideas within the path of the project.

4 | Project Plan

Reflections:

To me the Project plan hade many difficult parts. The first was to try to understand what I was supposed to write under the different titles. I hope I got at least most parts ok. Next issue was that I realized that some of my different sections have very similar content which might be wrong, I tried to fix it but didn't entirely succeed. My favorite part of the project plan was dividing the assignments in the iterations to smaller tasks and try to find a good balance in size and number.

4.1 Introduction

Hangman is a console application where the player guesses a word by suggesting letter after letter. The player shall be greeted when starting the game and when beginning her or she is presented with a word from a predefined list of nouns that is randomly picked. The number of letters is displayed with equally many underscore signs and a short clue connected to the word. The number of wrong guesses that the player can have is depending on how many parts the man to be hanged is drawn with. On every wrong guess a part of the hanged man is drawn. The hanging man is build using the available characters on the keyboard or by using Unicode characters. If to many wrong guesses are made the man is hanged and the game is over. If the player guesses right and the letter is included in the word the underscore sign/signs where the letter is located is replaced with the guessed letter. If the player can guess the right word he/she has the possibility to get into the high score list.

This project shall be executed in four iterations described in more detail in section five. The final application and last iteration is to be done week 12.

4.2 Justification

This application should be made for learning purposes. It will teach the student the process of software engineering, how to plan and execute a project through process and planning, modelling, software design and software testing.

4.3 Stakeholders

The end user/users, main client/clients, project manager and developers.

4.4 Resources

Time (9 weeks), literature; Software Engineering by Ian Sommerville, training by tutoring, lections sessions and QA: s, IDE; Visual Studio Code. Node.js and npm.

4.5 Hard- and Software Requirements

Software: IDE used is visual studio code, Node version 10.9.0 and npm version 6.2.0.

Hardware: Processor used during development is Intel(R) Core™ i7-8750H.

4.6 Overall Project Schedule

Iteration	Deadline
1	Friday, 8th of February 2019, 23:55
2	Thursday, 21st of February 2019, 12:00
3	Friday, 8th of Mars 2019, 23:55
4	This is the final product, v.12.

4.7 Scope, Constraints and Assumptions

Hangman is developed on the Node.js platform as a console application. The basics of the application is that it shall contain functionality for picking words from a predefined list of nouns. Displaying the words to the player with underscore signs representing every letter in the word. The player shall be able to guess letter after letter. If the letter guessed exists in the word the underscore sign is exchanged for the letter else a part of the hangman is drawn. The hangman is build using the available characters on the keyboard. If every part of the hangman is drawn the game is over and the player should be able to start over.

Extra features are a high score counting guesses and timer for counting down the time the player has to guess a letter.

The project is limited to 9 weeks and the knowledge and skills of one developer. There must be thorough documentations; vision, project plan and risk analysis before starting to implement functionality. And the application must contain the basic functionality when done.

It's assumed that the user is comfortable with console applications and has access to roughly the same software and hardware that was used in the development for this application.

5 | Iterations

The estimated time for each task is documented in parentheses after the task description and also in the time log table (7.1).

5.1 Iteration 1

1. Create a time log document and estimate the time for the tasks to be done. (30 min)
2. Create and document a vision for the project. (60 min)
3. Complete and document the project plan with everything known at the time being, using the project template as a skeleton for it. (3 h)
4. Create a risk analyze document. (2h)
5. Create a GitHub repository with README.md, package.json and .gitignore files for the project. (20 min)
6. Create the folders and modules needed for the project and organize the structure. Validate functionality by console.log (‘Hello World’) when running the application. (30 min)
7. Add npm packages for dev dependencies, standard and snazzy. (15 min)
8. Complement the time log with the actual time for the task and analyze the result. (1 h)

5.2 Iteration 2

In this iteration implement game logic for the basic features of the application.

Functionality for starting the application, randomly pick a word to be guessed displayed as underscore signs instead of letters. Checking if the letter added is in the word and either replace the undersign with the letter or start drawing the hanged man and functionality for win or lose.

Modell features using UML.

Then add features to the application.

5.3 Iteration 3

Plan for testing, perform and document the test.

Additional feature adding a high score list.

5.4 Iteration 4

Additional feature adding a timer that counts down the time the player has to guess a letter.

The complete application shall be done.

6 | Risk Analysis

Reflections:

It’s quite easy finding risks but judging the probability and impact is according to me more difficult. I’m still not quite sure if I’ve labeled every risk right. And I’m not sure if I should have more risks listed but I didn’t want to list risks just for the sake of making a long list. When it came to strategies I also had a problem, it felt ok coming up with strategies but how do I know that the strategies I came up with will work? Should one have back up strategies for the strategies?

6.1 List of risks

Risk	Probability	Impact
The size of the software is underestimated.	High	Tolerable
Key staff are ill during critical times in the project.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
Staff does not have the skills required	Moderate	Catastrophic
Required training for staff is not available	Very Low	Serious
Faults in reusable software components	High	Tolerable
Changes to requirements that require major rework.	Moderate	Serious
The rate of defect repair is underestimated	Moderate	Tolerable

6.2 Strategies

Risk	**Strategy**
Size underestimated	Careful planning for each task. Time estimates for every task complemented with the actual time for improving ability to estimate.
Staff ill.	Schedule and plan every task to reduce overtime and stress and if staff get ill se over the schedule and tasks and make a new plan.
Time underestimated	Schedule every task, estimate the time and add time to the estimate for eventual problems.
Staff lacks in skills	Read the literature available, attend the lectures, sessions and tutoring.
Requirements Changes	Iterative planning, documentation and implementation.
Repair underestimated	Add extra time to the estimated time.

Time log

7.1 Time Log

Iteration 1

Task	Time estimate	Actual time
1. Time log, Estimates.	30 min	3 h
2. Vision	60 min	2 h
3. Project Plan	3 h	10 h
4. Risk Analyze	2 h	5 h
5. Github repo, initial commit.	20 min	15 min
6. Folders, skeleton.	30 min	30 min
7. npm	15 min	20 min
8. Time log, actual + analysis.	1 h	45 min

7.2 Analyzes

Iteration 1

For the first iteration the actual time is unfortunately not as accurate as it should be. That is because of bad planning in the beginning, starting on multiple things at the same time and forgetting to clock the reading and learning part. Trying to minimize the damage the overall time for learning is divided and added to the tasks it's affecting. It became clear that the time needed for learning was very underestimated and will be more considered ahead.