Software Testing Documentation

Lars Cordewener
Software Engr.
Department CS
VU University

Arthur de Fluiter
OS and Security
Department CS
VU University

Radu Jica
Big data Engr
Department CS
VU University

Roy Overbeek
Foundations
Department CS
VU University

March 5, 2017

Abstract

A living document describing the creation and testing of a software project. This document describes a system made for a university course, ranging from the requirement engineering, to the testing of said system. Importantly this document will focus on the rationale for these particular tests, the benefits of choosing these particular tests, and the eventual results that were discovered.

The specific assignment was to create a simple software product and document it. In this particular case the end result should implement the game 'hangman', in which one player has to guess a certain word by picking letters they think might be in the word they are trying to guess. If the chosen letter is in the world it will be revealed on the location it is in the word and the player gets to guess again. The player only has an amount of guesses until a simple representation of a man hung from the gallows is shown on screen.

This document contains the testing from both the black box and white box style, however the latter will be added as the document develops over the weeks.

Introduction

For the course Software Testing (400439) as taught at the VU Amsterdam this document describes the requirements engineering and testing based on the material presented during the course. For this assignment each group was required to pick one of two assignments. These projects were developing a heart monitoring system or implementing a version of the game 'hangman'. For each assignment the groups were required to follow a particular development model, document the requirement engineering (and model accordingly), and perform a series of tests in both black box and white box style.

This group chose to implement the game of hangman, in which a player ultimately has to guess a word. The player is presented with an indication of how many letters the word has, and has the ability to both guess what the word can be, or which letter is in the word they are trying to choose. If the player guesses wrong the game builds an image of a hanged man piece by piece, however if the player guesses a letter correctly it will be added and shown in the place that the letter has in the word, which can be more than one.

The restrictions of the course show that it is not allowed to use GUI elements to implement the assignment, so the visualization is minimalistic. However the implementation still allows for all functionalities. Another restriction was the development model that each group should implement during the creation and testing of their product. Here the waterfall model was used, opposed to other implementations such as agile, or Test-Driven Development [1].

This model drives the design to first create a set of requirements. Based on these the actual design is made, after which this design is implemented. After the implementation comes the verification, which in this case comes down to testing the software in various ways. And lastly the model shows maintenance, which for this paper would come down to adjusting the implementation and verifying again to make sure the maintenance was successful.

Based on this assignment a set of requirements were generated, describing both the functionalities that the system should provide, as well as the quality standards that is should abide. These requirements would later be used to test the implementation as well as to present to another group for them to test the final implementation without access to the source.

The two different styles of testing as used for this system, namely black box and white box, differ greatly. White box testing is done with all the knowledge of the system, the documentation, and even the source code readily available. This allows the tester to verify certain qualities and requirements based on the implementation of the code.

Black box testing on the other hand is done with only the final implementation of the program and the requirements. This forces the tester to make assumptions

on how the code is implementation, meaning that the tests take on a different shape than the perfectly suited style that happens when using white box testing.

Both versions of testing can lead to different insights on the program and possibly faults it might have, and for each style of testing different methods will be used and described in the corresponding sections.

Requirements

The implementation of 'hangman' has certain quality standards that it should abide by, however a proper implementation also has to allow for, and present certain functionalities to the player. The following section will detail the different functional requirements, as shown in 1, and quality requirements, as shown in 2, for the implementation presented in this document.

ID	**Requirement**	**Description**
FR_01	Guess word	The user must be able to guess the complete word of the current game.
FR_02	Guess letter	The user must be able to guess a single letter they suspect is in the word.
FR_03	Display word	The system must display the amount of letters in the current word, as well as any letter that has been guessed correctly in the current game.
FR_04	Display guessed letters	The system must present the user with a list of letters that have already been chosen in the current game.
FR_05	Display guessed word	The system must show the user a list of words that have already been guessed in the current game.
FR_06	Check word for letter	The system must check the current word for a chosen letter and display that letter as part of the word if and only if chosen correctly.
FR_07	Check correct word	The system must be able to check if the word the user is guessing matches the current word of the game.
FR_08	Track error count	The system must keep track of the amount of errors and display it to the user at all times.
FR_09	Accepted letters	The system must allow for the user to input only a-z letters, with guesses being case-insensitive.
FR_10	No repeat guesses	The user should not be allowed to guess the same letter or word twice during a game, the system should not allow the input and instead let the user pick something else to guess.
FR_11	Correct length guesses	While guesses the complete word the user should only be allowed to guess words of the correct length (as shown) and with the letters shown in the right places.

Table 1: Functional Requirements

ID	Requirement	Description
QR_01	Ease of use	It should be easy to use, with only the use of a terminal/bash.
QR_02	Correctness	While checking the word for guesses the system should not make any mistakes in allowing or failing letters that should be in the word or the correct guess of the full word.
QR_03	Input	The system should not allow words to contain symbols/letters that the user is not able to choose or guess.
QR_04	Speed	Due to the simple nature, the system should not take long picking a new word or checking each choice of letter or guessed word.
QR_05	Game length	the system should ensure that each game allows for the same amount of mistakes.
QR_06	Presentation	The system should, at all times, correctly display the current state of the game. This includes the chosen letters, words, current word, and representation of the amount of errors.

Table 2: Quality Requirements

As these tables show the different requirements are rather elementary, however they are all important to ensure that the system operates without fail and provides the user with the experience they are looking for. Without these requirements or quality assurance the implementation would lack certain aspects.

Interesting to note however are the possible improvements that could be made to this setup. For example, this implementation could be extended to allow for a 'multiplayer' experience. This would allow some user to put in a word and then passing the keyboard to another user they would now be presented with the controls to guess the previously input word.

However since this is outside of the scope of this project these functions are not included.

Roy's changes

Definitions:

1. The *alphabet* consists of exactly the letters contained in the range a-Z.

2. A *word* is a sequence of non-whitespace characters with a length of at least 2 and at most 45 characters.

3. A *word guess* is a word entered by the user on the command prompt, possibly surrounded by leading and trailing whitespace characters.

4. A *letter guess* is a single character entered by the user on the command prompt, possibly surrounded by leading and trailing whitespace characters.

5. A *guess* is either a word guess or a letter guess.

6. The *stripped guess associated with a guess* x is the result of removing all leading and trailing whitespace from x.

7. The *secret* is the word selected by the system for the user to guess.

8. The *partially revealed secret* is the secret in which 0 or more positions have been hidden from the user.

9. A word guess is *consistent* with the partially revealed secret iff the following propositions hold:

    - the length of the associated stripped guess matches the length of the secret;
    - every revealed character at position i in the partially revealed secret is also at position i of the associated stripped guess.

10. Let x be the associated stripped guess of a letter guess y. Then y is *valid* iff the following propositions hold:

    - x is in the alphabet;
    - x is not in the list of previously guessed letters.

11. Let x be the associated stripped guess of a word guess y. Then y is *valid* iff the following propositions hold:

    - all the characters in x are in the alphabet;
    - x is not in the list of previously guessed words;
    - x is consistent with the partially revealed secret.

ID	Requirement	Description
FR_01	Guess word	The user must be able to enter a word guess.
FR_02	Guess letter	The user must be able to enter a letter guess.
FR_03	Display partial secret	While a game is in progress, the system must at all times display a partially revealed secret.
FR_04	Display guessed letters	While a game is in progress, the system must at all times display a list of guessed letters.
FR_05	Display guessed word	While a game is in progress, the system must at all times display a list of guessed words.
FR_05	Display hangman	While a game is in progress, the system must at all times display the hangman.
FR_05	Partial secret init	The partially revealed word initially hides all positions of the secret.
FR_05	Guessed letter init	The list of guessed letters is initially empty.
FR_05	Guessed word init	The list of guessed words is initially empty.
FR_05	Hangman init	The hangman is initially empty.
FR_06	Letter guess 1	If a valid letter guess c is made, and c is among the hidden positions of the revealed word, then the system should reveal all positions that contain the character c.
FR_06	Letter guess 2	If a valid letter guess c is made, and c is not among the hidden positions of the revealed word, then the system should extend the hangman.
FR_06	Letter guess 3	If a valid letter guess c is made, then c should be added to the list of guessed letters.
FR_06	Letter guess 4	If an invalid letter guess c is made, then the user should get feedback on exactly one violated property (see definitions), and be prompted to try again.
FR_06	Word guess 1	If a valid word guess w is made, and w matches the secret, then the system should reveal all positions of the partially revealed secret.
FR_06	Word guess 2	If a valid word guess w is made, and w does not match the secret, then the system should extend the hangman.
FR_06	Word guess 3	If a valid word guess w is made, then w should be added to the list of guessed words.
FR_06	Word guess 4	If an invalid word guess w is made, then the user should get feedback on exactly one violated property (see definitions), and be prompted to try again.
FR_06	Termination 1	If the partially revealed secret contains 0 hidden characters, then the game ends and the player wins.
FR_06	Termination 2	If the hangman is complete, then the game ends and the player loses.
FR_06	Play again	If the game ends, then the player is asked whether they want to play again.

Table 3: Functional Requirements

Testing

This chapter will detail the different tests that were performed on the implementation. Additionally it will also describe what these tests entail and the specifications for this system. Furthermore this section will be divided into a black box and white box section, since each style has different methods of testing.

Black box testing

White box testing

References

[1] S. Balaji and M. S. Murugaiyan, "Waterfall vs. v-model vs. agile: A comparative study on sdlc", *International Journal of Information Technology and Business Management*, vol. 2, no. 1, pp. 26–30, 2012.