TÉLÉCOM SUDPARIS - INSTITUT MINES-TÉLÉCOM

Word Search Puzzle - Report

Oscar GUILLÉN and Patricia REINOSO

February 17, 2017

1 INTRODUCTION

Advances in the computer science and the increase of the market in technology, the software development has become a powerful branch that permits the improvement of thousands of corporations around the world. The migration to the software development as a result of the pursuit of automation, and its increase as a result of the globalization, lead the software engineering as an important discipline which has permitted the increase and management of the software development.

The aim of this project was to develop a simple software application using a systematic method to develop and manage the entire software development. The project consisted on the creation a Word Search Puzzle game, where the player finds hidden words on a not-static board full of letters; applying the basics of software engineering, during the development even on a small scale.

The objective of this report is to explain the entire process of development of the project using the software engineering basis. First, the process life-cycle used for the development is presented, detailing the method election and a description of each phase fulfilled.

Second, the composition of the team and the approach of distributing tasks and responsibilities among the team of development is described. Subsequently, the tools used to facilitate the process of development and the programing language selected are presented. In addition, development aspects such as debugging, reusing code, documentation, version control and some implementation decisions are explained. Finally, the conclusions and

some recommendations and feedbacks collected during the process of development, and references are shown.

2 SOFTWARE PROCESS LIFE CYCLE

The project style used intended to make an arduous analysis of requirements and design. Also, the good task distribution and responsibilities distribution among the members of the team and the tracking of time per activities were important into the project style. Given the project style and the working features of the members of the team, the waterfall model of development fit perfectly for this project.

Initially, a waterfall model was used. The problem was analyzed and the requirements established. Then, the design was started, where all the methods and functions necessary to code the game according to the requirements were defined. Nevertheless, the expectations were too big for the time available, and there would not be enough time to develop everything, or at least it would not be the purpose of the course. Therefore, a different life cycle process was started: Prototype Evolution Model.

An initial set of requirements were selected from the one already specified. Those requirements were used to design, build and test a simplified prototype of the game, where the basic functions were implemented. Allowing, to complete a playable first version of the game. The description of each phase fulfilled is presented below:

2.1 ANALYSIS

Word Search Puzzle is a game that have been widely implemented. This project attempts to create a dynamic version of the game, that allows to slide the rows horizontally and the columns vertically. The difficulty of the game should be high enough so that the it represents a challenge for the players, but not too high, so that they do not get frustrated.

2.2 REQUIREMENTS

To create the requirements of the game, it was necessary to imagine what a customers of the product would want to achieve. Therefore, to specify the requirements, 2 documents were created:

- User manual: includes the game rules from the point of view of the user.
- Engineers manual: document including all the requirements, destined to the engineers.

These 2 documents complement each other and are consistent.

2.3 DESIGN

The structure of the software were specified by the creation of 3 instruments:

- User Flowchart: describes the logic of the game from the point of view of the user. It helps to define the functions and methods required to implement the game.
- Functional Decomposition Independency Graph: describes all the functions and methods that are going to be coded on the implementation phase. It includes the relations among them and the parameters needed.
- Skeleton of the code: includes the structure of the code. It is written in python and uses its data structures. This file matches the design specified on the functional decomposition independency graph to the object-oriented programming in this language.

2.4 IMPLEMENTATION

The features were divided into tasks and, these tasks were divided into responsibilities which were assigned to the members of the team. The tasks were created from further inside until the more general. Thus, some responsibilities needed to be completed before starting more general ones. The functional decomposition independency graph was used to identify which tasks are more general than others and the initial design of the functions were used to start writing the functionalities adding new ones if was necessary.

The main functionalities were divided into 4 parts:

- Main general functions: those functions that are used in the main but which do not use more functionalities.
- Dictionary importation functions: which build the dictionary of the game, importing some files with a specific structure. This functionality used other functionalities which were developed before its implementation.
- Setup function: which prepare the entire configuration before the game. It selects the list of words that the user needs to find and build the board according to the requirements. Makes everything ready so that the player can start playing the game.
- Start game function: which was divided into functionalities to atomize the function of each part and finally build the entire logic of the game. It manages the turns, gets and executes the commands of the users.

Finally, the parts above were assembled into one general function: Main function.

2.5 TESTING

Only validation test were executed. The process of testing consisted on playing the game, checking each flow of the game. Initially the regular flow of the game was follow, and then, the input of invalid values and different scenarios where the program may fail were proved.

There were not found much bugs during the testing process. The good documentation has helped fixing those errors.

3 TEAM

This was a small project, therefore the team needed was 'Individual' (2 people). Given that on small teams, individual abilities are crucial for the success of the project, the selection of the programming language was truly influenced by the fact that both members of the team had a little experience working with the programming language selected.

Both member had many roles on the team. All of them worked on the different phases of the development of the software, in order to discuss all the options, solve doubts, make decisions and learn. The fact of sharing same language and background on the academic domain, made the co-working and the understanding very easy.

4 TASKS PLANNING

First, the members worked on the requirements at the same time, searching the best and the strongest specification of the requirements. The initial requirements of the game that we created were too ambitious. Therefore, the creation a first version of the game, and changing of life-cycle mode, were needed. The description of the tasks process for the version 1 is described below:

A subset of the requirements was selected. Subsequently, the members worked on different diagram of design. One was in charge of the User Flowchart and the other one of the Functional Decomposition Independency Graph.

The coding was divided on tasks, dividing the implementation of the functions between the members of the team. It was necessary to maintain a partial division of the features among the responsibilities of each member trying to reach the faster way for the implementation. Then, the members worked on the main function together, assembling the features as a whole.

Finally, the program was tested, adding the feedback and the results of the testing. It was important to test each functionality developed by the other member, trying to find bugs on of the other functions instead of the own functions. Moreover, when a member found a bug, it was fixed immediately.

During the development of the project, the expected time and the actual time that that each task took were specified. Those values are presented below:

Table 4.1: Task planning table.

Task	Specific task	Responsible	Expected time	Actual time
Requirements	Engineers document	Patricia	1 hour	2 hours
	User guide	Patricia	1 hour	2 hours
Design	Functional decomposition independent graph	Oscar	2 hours	7 hours
	Flow chart user	Patricia	1 hour	2 hours
Implementation	Main general functions	Oscar	10 hours	18 hours
	Dictionary methods	Oscar		
	Setup functions	Oscar - Patricia		
	Startgame functions	Oscar - Patricia		
	Instructions methods	Patricia		
	Integration - Main methods	Oscar - Patricia		
Testing	Validation tests	Oscar - Patricia	2 hours	2 hours
	Fixing bugs	Oscar	2 hours	3 hours
Documentation	HTML generation	Patricia	2 hours	3 hours
	Text file generation	Patricia		
	Code export to pdf	Patricia		
	Include comments while programming	Oscar - Patricia	-	-
	Report	Oscar - Patricia	3 hours	6 hours
Total			24 hours	45 hours

On the table, the distribution of the tasks between the members of the team is presented. Fairness on the distribution was accomplished. Each member worked approximately the same time. Nevertheless, the team failed calculating the time that each task would take. The difference between the expected time and the actual time taken is considerable.

This is the result of ignorance of the work rhythm of the members of the team, underestimation of the proportion of the project, unexpected bugs while programming, details missing from the design and the requirements, tasks that were not considered at the beginning, and lack of experience using debuggers and documentation tools.

5 TOOL SUPPORT

In this project was used tools which were supporting the facilities of develop it.

1. Code2pdf: python package that converts source code into pdf file with syntax highlighting. [2]

2. Dia: an application used to create diagrams. It was used to create the Functional Decomposition Independency Graph and the User Flowchart.[1]

3. Git: a software used in software development for the version control of computer files and the use of those files for a workgroup. Some version control software depend on a central server that control the changes and the manage of files. Git uses the definition of a local repository which helps in term of independency, well it does not depend on a central server or a network access.

4. GitHub: web-based Git repository used to manage version control of the project.

5. Pydoc: python module that generates documentation. This module extract the comments (docstrings) written in the program and creates various types of files.[5]

6. Python: programming language used to develop the project.[6]

7. Python Debugger: feature of the programming language used to debug the program.[4]

6 PROGRAMMING LANGUAGE

The project was completely developed using Python, particularly the version 3.4. Python is an object-oriented, interpreted programming language, that combines power with very clear syntax and it is very easy to learn. Python was selected because, both members of the team had a little experience working with this language and because it is very easy to write. Python contains some aspects that helped the implementation of this project. Those aspect are the following:

- It manages the strings and lists easily.
- It contains a simple and clear syntax and semantic.
- It is multiparadigma. It allows the Imperative programming, that is a simple and coherent way for develop the game's logic. Also, it allows the Object-oriented programming, managing objects as classes. This implementation of the game is strongly based on classes.
- It provides tools which help the build and manage of a helpful documentation.
- It is open source software. Thus, it supports a community which is constantly developing and testing the implementation of python and a big amount of libraries. In this project some helpful libraries of python were used.

Nevertheless, Python contains some aspects which made it difficult. Those are:

- It does not make the difference among types. Very fragile while passing parameters. Thus, it was necessary to create a big amount of verifications, and assure that all the types were consistent.

7 VERSION CONTROL

To manage the changes on documents and the evolution of the prototype, version control was used. In particular, free and open source distributed version control system, Git. This tool allows to create subversions, multiple branches and parallel develop of each phase of the project. The project is stored on a GitHub repository. [9]

7.1 GIT WORKFLOW

Three main branches were created:

- master: stores the official release of the project. It is the principal branch. It contains a correct and tested version of the project.
- develop: it was the integration branch for the requirements and the design that were originally created. When we decided to change of process life-cycle, it became an integration branch for subversions. It lies from the master branch.
- version1: is the integration branch for features developed for the first version of the project. It lies from the develop branch.

Several branches were additionally created to develop the features of the game.

The following branches, lie from develop branch.

- requirements: was used to add the requirements of the entire project. Includes documentation. Then, the requirements were refined and simplified for the first version
- design: was used to develop the first design of the project. It includes documentation and source code. Then, the design were refined and simplified for the first version

The following branches lie from develop branch.

- design-v1: was used to develop the documentation and source code of the design for the first version.
- requirements-v1: was used to refined the requirements as a result of the development of the first version.
- dictionary: was used to develop the feature of the dictionary of word of the game.
- setup: was used to develop the feature of setting up the game.
- turns: was used to develop the feature of turns per players into the game.
- read_command: was used to develop the feature of reading a command from the terminal through the game.

- find-word: was used to develop the feature of finding a word into the clues of the game.
- mainFunction: was used to assemble all the feature of the game and create the main function.
- pydoc_doc: was used to develop the visualization of the documentation of the project into html or text.

The following branches lie from develop.

- bugs: was used to fix general bugs found into the program.
- menu-bugs: was used to fix some bugs found into the program which were associated with the menus in the program execution.
- import-bugs: was used to fix some bugs found into the program which were associated with importations of data in the program execution.

8 FILES

The files that can be found on the repository are presented below:

- WordSeachPuzzle/design : directory including all the files related to the design of the project.
  - design.py : initial skeleton of the code.
  - flow_chart_v1.dia : source code of the diagram of the flow-chart user.
  - flow_chart_v1.png : flow-chart of the version 1 of the game in .png format.
  - functional_decomposition_independency_graph.dia : source code of the diagram of functional decomposition independency graph of the version 1 of the game.
  - functional_decomposition_independency_graph.png : diagram of functional decomposition independency graph of the version 1 of the game in .png format.
- WordSeachPuzzle/documentation : directory including all the files related to the design of the project.
  - code_puzzle.pdf : source code of the project exported to pdf.
  - documentation.html : documentation of the code in .html format.
  - documentation.txt : documentation of the code in .txt format.
- WordSeachPuzzle/makefile : makefile created to execute the game easily.
- WordSeachPuzzle/puzzle.py : source code of the project.

- WordSeachPuzzle/README.md : description of the project. Includes how to execute the game and the rules.
- WordSeachPuzzle/requirements : directory that contains the requirements of the project.
  - planning-v1.odt : file that contains the task distribution and the expectation of time for each task.
  - requirements.odt : file containing the requirements of the game. It is destined to the engineers.
  - user_guide.odt : user guide including the rules of the game.
- WordSeachPuzzle/rules : directory that includes the rules of the game.
  - instructions.html : html version of the rules of the game.
  - instructions.txt : plain text file that is imported and used during the game.
- WordSeachPuzzle/subjects : directory that contains all the subjects included on the game.
  - animals.xml : contains a list of animals. It is a xml file following an specific format that is imported during the game.

9 DEBUGGING

A first analysis was done after the testing process. This analysis was done using the strong documentation of the code and trying to identify errors without the use of a debugger. Nevertheless, the Python Debugger pdb was used as a debugger, particularly to find problems at the moment of building the board of the game, due to the fact that it was taking too much time. The debugger is available as a source library of python.[4]

10 REUSING CODE

Reusing code is an useful approach that could save a lot of work during the software development process. This project contain some reused code from the python source libraries. First, the use of the 'os' library to manage the system files and directories. Second, the use of the xml parsing library to manage .xml files with python. Finally, the use of the re library to manage the parsing of regular expressions. The fact that the reused code comes from library, give confidence to properly implementation and testing of the code. Those libraries are available into the python documentation.

11 DOCUMENTATION

The documentation of the code was generated, using the module Pydoc. Using this tool, 2 files were created: 'documentation.html' and 'documentation.txt'. These 2 files contain the description of each class and, the parameters and return type values of each method and function on the source file of the project. In addition, a pdf file that includes the source file with highlighting was created.

Even if there exist several tools that provide similar functionalities, Pydoc was selected because it was really easy to install and to use. It is important to mention that installing Docutils, Doxygen and Sphinx was tried with no success.

Good code documentation permits to avoid ambiguities on the code. Documentation is very important when programming using python due to the fact that this language does not do an explicit declaration of variables, types can be completed mixed, generating inconsistencies on the code. In this project, the documentation of parameters and returns were partially useful.

In addition, good documentation improves the quality of the software and speeds the implementation. It is extremely useful in case of extension, maintenance and reuse of the code. Moreover, the activity of identify errors and bugs needs an arduous work which sometimes requires an important study of the source code. A correct and complete documentation allows the possibilities of study faster and more efficient the source code of a project, reducing the complexity of finding problems. In addition, a standardization of the documentation permits to use powerful documentation tools as pydoc, used in this project.

12 IMPLEMENTATION DECISIONS

12.1 SUBJECTS AND DICTIONARY

The subject is a class defined in the project. This abstraction permits the possibility to add subject with its name and words as its content. The object is added into a dictionary which is another abstraction into the project design and implementation.

The information of the subject is supplied into special files which are XML files. Those XML files contain a specific syntax which supplies the name of the subject and the words into the subjects as follow:

```
.       <data>
.           <subject name="subject">
.               <word>word 1</word>
.                       ...
.               <word>word n</word>
.           </subject>
.       </data>
```

The non-alphabetic strings are not allowed. Thus, they are ignored and are not added into the dictionary. It is important to indicate that it exists only one file per subject. If there are more than one subject into one file, only the first subject is going to be added. In addition, any file is added into the directory WordSearchPuzzle/subjects. Finally, if there is an error or there are added no words, the program is going to interrupt the execution.

12.2 BUILDING THE BOARD

The board is a class defined into the project. This abstraction permits an approach to encapsulate the definition of the board. The construction of the board has lead to implement an algorithm partially random to permit the fairness among the several games.

The words were put randomly into the board from the biggest to the smallest one, taking always a random decision. First, the algorithm decides randomly whether put into a column or into a row. Second, selects randomly the row o column where it could be added. Finally, decides a position into the row or column to trying the addition into the board. If it was not added, the process is repeated until adding the word successfully.

However, this algorithm needs to be fixed after testing and debugging (this case was accomplished to the debugger tool, pdb). This is a bug that was not fixed completely, it was reduced decreasing the number of words into a board (from 12 to 10). The random behavior is possibly the most fair, but it is uncontrollable. Thus, it could lead to cases where the state of the board does not allow to successfully all the words in an appropriate amount of time.

The algorithms can be improved using a different approaches. In addition the the state of the board that is presented to the user could be more scramble, rotating it more, so that it represents a bigger challenge to the user finding the words.

13 CONCLUSIONS

Waterfall process life cycle model is a methodology that emphasizes the completion of each phase and targets early planning and design. Even if this model is widely used, and it was the initial approach selected for the development of this project, it did not fit completely to achieve the end initial requirements established on time.

Adopting a prototype evolution model, allowed to complete each phase of the the process on time. This model improved the productivity and allowed to deliver a working system considerable earlier.

Specification of the requirements and the design is a very difficult task and they are very important, in order to accomplished successfully the implementation of the software. However, it is not easy at all to capture all the details at one time. During the coding, some important functions and not specified elements were found, so it was necessary to return to the requirements and the design to resolve the aspects missing. Only practicing and studying it is possible to improve the ability to capture requirements and to design properly a software.

Besides, for software engineers, it is really important to calculate the most accurately, the amount of time that a completing a task would take and not to be afraid of being sincere about it. Measuring the time while working is the way of learning the working rhythm of a person, this should be an habit for software engineers. In addition this is the way to make closer the expected time and the actual time taken.

Selecting the programming language and the tools can facilitate the development process. This selection has to be done considering the characteristics of the problem and the abilities of the members of the team, especially on small teams.

Including proper documentation of the code is crucial and always useful. Inconsistencies can be easily found thanks to the documentation. This is a process that must not be done at the end of the project.

To reuse code can facilitate implementation, but it is necessary to make sure that the code works properly. In addition, it is important to always acknowledge other people's work.

In conclusion, from the development of the game, aspects the process life cycle and proper aspects of software engineering were learnt. These elements need to be strengthen, but what was learnt will be used and practiced in future projects in the academic and in the professional domain.

14 RECOMMENDATIONS

Designing a User Flowchart before the Functional Decomposition Independency Graph is very helpful. The flowchart allows to clarify the purpose and the sequence of events. It also helps to have an idea of the dependency of the functionalities and theirs order, permitting to think what function is more general than others.

The correct design will make the implementation easier and faster. The implementation phase would have taken longer if the design would not have been made very carefully. Even if in occasions some elements missing were found. Thereby, a careful design or even better, a well-done design implies a friendly implementation.

In the implementation process, it is important to take in consideration a constant and standard documentation. It was not only used to understand the code but also to find the errors into the process of debugging.

It is important not to underestimate the time of the tasks and to adapt the requirements can be successfully finished on the given time. Otherwise, changing strategies and process life cycle would be necessary.

REFERENCES

[1] Apps/Dia - GNOME Wiki. Available :
`https://wiki.gnome.org/Apps/Dia`

[2] Code2pdf package documentation. Available:
`https://pypi.python.org/pypi/Code2pdf/0.0.1`

[3] Gibson, J. (2017). Software Process Life Cycle. Available:
`http://www-public.tem-tsp.eu/ gibson/Teaching/CSC7003/L10-Process.pdf`

[4] Python debugger pdb documentation. Available:
`https://docs.python.org/3.4/library/pdb.html`

[5] Pydoc documentation. Available:
`https://docs.python.org/3/library/pydoc.html`

[6] Python Official Documentation. Available:
`https://docs.python.org/3/l`

[7] Python Wiki. Available:
`https://wiki.python.org/moin/`

[8] Raskin, J. (2005). Comments Are More Important Than Code. Available:
`http://www-public.tem-tsp.eu/ gibson/Teaching/Teaching-ReadingMaterial/Raskin05.pd`

[9] Word Search Puzzle. Project Repository:
`https://github.com/OssyGuillen/WordSearchPuzzle`