

Gamified Acquisition of Software Feedback
Master's Thesis in Game and Media Technology

ICA - 3935361

Nick Linakis
Department of Information
and Computing Sciences,
University of Utrecht,
Utrecht, The Netherlands

Supervisors: Dr. Fabiano Dalpiaz, Prof. dr. Johan Jeuring

`n.linakis@students.uu.nl`

August 31, 2015

Acknowledgments

First of all I would like to thank Fabiano Dalpiaz for supervising this thesis project, and providing me with most helpful insights on this subject. Also, I would like to thank Raian Ali, Malik Almaliki and Alimohammad Shahri from Bournemouth University who under the collaboration of Utrecht and Bournemouth Universities (BUUU) they provided valuable data and acumen in order to conduct my research. Finally, I owe a debt of gratitude to everyone who were kind enough to take some time for my research, so that they could fill my database with user-feedback measurements. My sincere thanks to all of you!

I hope someday to write something worth plagiarizing.

Unknow Author



Abstract

Efficient and successful software maintenance and evolution processes could lead to better quality software applications, which could meet the end-user's requirements and preferences. A decisive contribution to software maintenance and evolution could be achieved by defining more adequate and effective feedback acquisition practices in combination with the use of gamification principles. The overall objective of this study is to create and validate a feedback acquisition framework which will be embedded in already existing applications and will engage and motivate the users by exploiting the advantages of gamification elements. To achieve this, we realized a conceptual solution by performing literature search and assessing current feedback acquisition practices. Furthermore, with the implementation of the most important functional requirements of such a framework. Last but not least, we validated the usability and effectiveness of our framework by executing a controlled experiment between a treatment and a control group. Each group provided feedback remarks by using our gamified feedback acquisition mechanism and an existing feedback acquisition tool respectively. Experiment results and analysis indicate that our developed tool appeared to be more usable and more effective on acquiring feedback remarks, compared to the existing feedback acquisition mechanism used for the experiment.

Contents

Acknowledgment	i
Abstract	ii
1 Introduction	1
1.1 Background	1
1.2 Problem Definition	2
1.3 Research Objectives and Research Question	3
1.4 Solution Approach Overview	4
1.5 Scientific and Societal Relevance	6
1.6 Document Structure	7
2 Related Work	9
2.1 Software Evolution and Maintenance	9
2.2 User Involvement in Software Evolution	14
2.3 Gamification	17
3 Approach and Design	21
3.1 Problem Domain	21
3.2 Conceptual Framework Requirements	22
3.3 Conceptual Solution Outline	23
4 Framework Implementation	28
4.1 Generic Implementation Details	28
4.2 Framework Design	28
4.3 Framework Architecture	35
5 Experiment	38
5.1 Experiment Scope	38
5.2 Experiment Design	39
5.2.1 Design Choices	39
5.2.2 Participant Selection	40

5.3	Experiment Process	40
5.3.1	Tool Preparation	41
5.3.2	Scenario Preparation	45
5.3.3	Survey Preparation	46
5.3.4	Pilot Experiment	47
5.3.5	Final Experiment	48
5.3.6	Analysis	48
5.4	Threats to validity	49
6	Results	51
6.1	SUS Results	52
6.1.1	Embedded Gamified Feedback Tool	52
6.1.2	Embedded Mantis Bug Tracker Tool	55
6.2	Feedback Remarks Acquired	58
6.3	Discussion	59
6.3.1	Framework Usability	60
6.3.2	Feedback Remarks Acquired	62
7	Conclusions, Limitations, Future Perspectives	64
7.1	Answers to the Research Questions	64
7.2	Limitations	65
7.3	Future Work	66
A	Appendix	67
	Bibliography	83

Chapter 1

Introduction

1.1 Background

A software application is defined as a packaged configuration of software components, or a software-based service with auxiliary materials, which is released for and traded in a specific market [33]. Software applications are designed and developed in order to address general use by end users or specific by a certain user or an organization. General purpose application software is a software type which is used for a variety of tasks and is not limited to a certain function. For example a text editor could be classified as general purpose software as it could allow a user to write a novel, create a restaurant menu or make a poster. Custom software (also known as bespoke software or tailor-made software) is software that is specially developed for some specific organization or other user. As such, it can be contrasted with the use of software packages developed for the mass market, such as commercial off-the-shelf (COTS) software, or existing free software.

Software production is the process of designing and developing a software application. It includes several processes and it is often considered as a subset of software systems development life cycle [18]. A decades-long still-to-be-attained goal has been to find repeatable and predictable processes that improve productivity and quality of software applications. In order to achieve better quality design and development of software applications, the maintenance and evolution of a software application should be properly managed. In the late 1970's, a widely cited survey study by Lientz and Swanson [34] revealed the very high fraction of life-cycle costs that were being dispensed on maintenance. They categorized maintenance activities into four classes:

- *Adaptive*: modifying the system to cope with changes in the software environment,

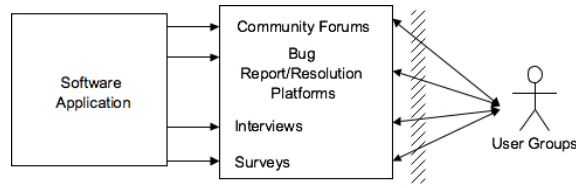
- *Perfective*: implementing new or changed user requirements which concern functional enhancements to the software,
- *Corrective*: diagnosing and fixing errors, possibly ones found by users,
- *Preventive*: increasing software maintainability or reliability to prevent problems in the future.

The survey showed that around 75% of the maintenance endeavor was on the first two types, and error correction consumed about 21%. A more recent study conducted by Koskinen [29] in 2003 showed that software costs devoted to system maintenance and evolution reach more than 90% of the total software costs. These evidence allow us to realize that software development organizations invest more and more on maintenance and evolution. A fact, which not only makes those two processes very valuable for software development development organizations, but also constructs a field worth investigating and improving.

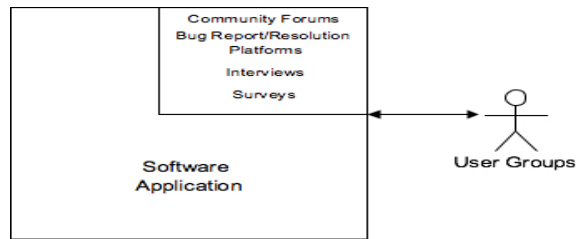
1.2 Problem Definition

Important, but somewhat troublesome conclusions as reported by [45] a decade later, is that maintenance problems are pretty much the same as during the 1970's (except for minor changes), despite improvements made in structured methodologies and techniques [31] [8]. In terms of specific problems, personnel effectiveness problems, i.e. skills, motivation and productivity and personnel problems of maintenance programmers, i.e. turnover and availability, have shown a rise, while problems concerning users' knowledge of computer systems have declined. Contribution of end user is crucial during the new requirement data gathering and analysis and that all software that is useful and successful stimulates user-generated requests for change and improvements [12].

These user-generated requests should be taken into account as a software application undergoes modification of code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software application in such a way, which meets user preferences while preserving its core functionality elements. Software maintenance is defined in IEEE Standard 1219 [23], as the modification of a software application after delivery to improve performance or other attributes, to correct faults, or to adapt the application to a modified environment. Software evolution is defined by [1] as "the dynamic behavior of programming systems as they are maintained and enhanced over their life times". The ability to change and evolve software easily, quickly and reliably is a "grand challenge" within software engineering [38]. Too much focus is currently on the technology and not on the end-user [36].



(a) Existing feedback acquisition process.



(b) Feedback acquisition process to be.

Figure 1.1: Feedback Acquisition Processes.

Remaining informed of the users' opinion on the role of software systems in order to meet their constantly changing needs is crucial. Especially in software applications used by a very large number of people who originate from different backgrounds. Solutions are going to be essential to meet the needs of businesses, where change is constant and urgent [31]. Thus, it is expected that software evolution will be positioned at the center of utmost importance in the field of software engineering in the future.

Users need stimuli that make them eager to contribute to the process. Current practices do not involve systematically the user in feedback acquisition process, neither provide the appropriate incentives that motivate and engage the public for providing their feedback. Solutions are going to be essential to meet the needs of businesses, where change is constant and urgent [31].

1.3 Research Objectives and Research Question

In this research, we are going to investigate how it is possible to progress from existing software feedback acquisition mechanisms (*Figure 1.1.a*), to a system where feedback acquisition is encapsulated in the software application (*Figure 1.1.b*). Current practices in software development are approaching groups of users by using external tools in order to gather the required

information which will allow for improved software application releases (e.g. discussion forums for feature requests, reports, bug resolution platforms for managing bug reports, etc.) [15].

Feedback acquisition mechanisms vary depending on the software application but they are all driven by the principle of obtaining users' wishes in order to deliver better quality software applications. We would like to integrate in a reusable manner, currently used feedback acquisition practices under the hood of already existing software applications. We are going to examine under the scope of software engineering and provide remarks to the following research question and sub questions:

- Main Research Question:
 - How to enable the development of software systems where feedback acquisition and processing are integral components of the system to-be?
- Sub Questions:
 - Which are the appropriate mechanisms for effective feedback acquisition?
 - How to integrate such feedback mechanisms in a reusable manner into existing software development practices?
 - How can we engage and involve more users in a systematic way?

In order to nettle users interest and involve them into software evolution we will delve into feedback acquisition by using gamification and crowdsourcing techniques. In addition, we are going to examine what the effects of the introduction of gamification in software feedback acquisition are, as well as the possibilities to maximize the participation of the users.

1.4 Solution Approach Overview

Current software feedback acquisition practices do not constitute integral components of existing software applications. Moreover, feedback gathering methods neither motivate or engage the end users in order to provided their remarks, which as a result leads to minimum feedback disclosure.

In order to address these problems, we are investigating how current processes on software maintenance and evolution are executed. All the basic knowledge needed regarding this research project will be acquired through literature review, including scientific papers, reports, and publications in scientific magazines of research institutes. Moreover, we will identify the gamification elements that will be appropriate to be used in order to motivate users to provide their feedback.

There is need for improvement in current software maintenance and evolution characteristics. In order to identify any practices which could contribute in the improvement of software maintenance and evolution, we are going to examine the contexts of gamification and crowdsourcing techniques under the scope of this research work. Crowdsourcing [9] is an emerging on line, distributed problem-solving and production model, with the use of which a problem is solved through the involvement of a large number of people. Crowdsourcing has the potential to be a suitable technique for large-scale user involvement. According to [11], crowdsourcing is “a strategic model to attract an interested, motivated crowd of individuals capable of providing solutions superior in quality and quantity to those that even traditional forms of business can”.

Moreover along with the context crowdsourcing, gamification could provide an important contribution to software maintenance and evolution. Brian Burke redefines gamification [10] as “the use of game mechanics and experience design to digitally engage and motivate people to achieve their goals”. According to Gartners speculations [10], more than 50% of organizations that manage innovation processes will have gamified those processes by 2015. The development and use of a software tool that includes gamification elements in order to involve people in the evolution of a software application may be able to better contribute to the quality of future software releases.

We go through the development and implementation of a software framework, which contains gamification, crowdsourcing elements together with current techniques that are being used to gather user feedback. We follow an IDE - plug-in development approach during the development process of our tool, by choosing a certain IDE and developing a plug-in which refactors existing software code and adds an enhanced feedback acquisition mechanism.

The properties of this mechanism will be based upon the literature study concerning current feedback acquisition techniques and the choice of applicable gamification components that will serve as user incentives. This approach ensures a better implementation structure which we expect to have an impact on many important software qualities such as enhanced re-usability and reduced complexity (*Figure 1.2*).

Upon the completion of the aforementioned step, the framework will be tested upon an existing software application (e.g. an open source software application) in order to identify the efficacy and performance of such approach.

In order to engage users and acquire as more responses as possible concerning the evaluation of a certain software application, we use a market-based approach described in [14]. According to this research, a new paradigm is proposed, which allows for the involvement of a large number of people in software evolution and also involves gamification elements in order to attract users attention and feedback.

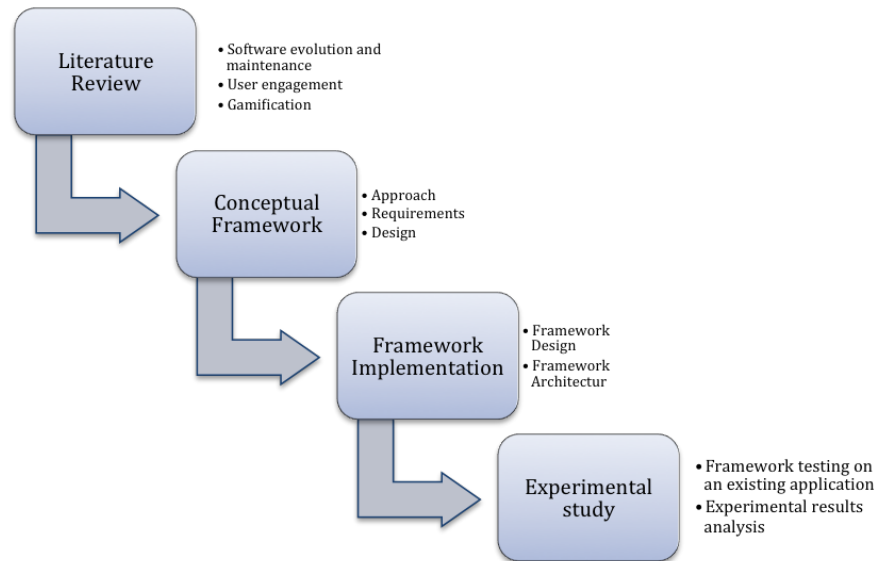


Figure 1.2: Research process overview.

1.5 Scientific and Societal Relevance

A decisive contribution to software maintenance and evolution could be achieved by defining more adequate and effective feedback acquisition practices in combination with the use of gamification principles. Gamification has been around for a long time, but its awareness has been documented the first time in 2008 [41]. While many organizations already use the principles of gamification, the trend towards a more fun environment in software engineering has just commenced [26]. This research provides a study on feedback acquisition along with the provision of incentives and motives to the end users as part of the gamification context. Furthermore, the framework design and evaluation demonstrated an example how these concepts can be combined in terms of scientific value. The results can be used for future research to deeper elaborate in this specific field of interest.

Gamification has the hidden potential to increase work and learning performance by changing people’s behavior in a positive manner. This is mostly

done by increasing engagement with the application of game-design elements [42]. This aspect could potentially encourage end users to participate more actively and efficiently in feedback acquisition procedures. Due to the fact that most software applications really depend on the software maintenance and evolution processes, it is very important to get the end users involved in these processes. With game design principles, end users can have more pleasure while providing their remarks concerning a certain software application and deliver valuable insights that otherwise would be missed.

Software companies can gain benefit from a concrete example of how effectively an embedded software gamified feedback acquisition mechanism was applied in an already existing software application. By observing the application of our tool and the results of this application, companies might try to apply similar approaches for their software applications in order to improve software maintenance and evolution processes in combination with motivating the end users in providing their feedback. With the combination of these aforementioned elements this research was a first step in this direction, resulting in valuable information that could be used and applied by researchers and professionals.

1.6 Document Structure

This work is structured as following, firstly in Chapter 2 we overview the background work on software evolution, software maintenance, user involvement in software development and the gamification principles.

In Chapter 3, we define the problem domain, we proceed by identifying a solution to the aforementioned problem statement and we provide the conceptual framework requirements and outline for our framework.

Furthermore, in Chapter 4, we provide with a thorough description of the implementation of our framework. We define the problem domain which this study addresses, we analyze the conceptual requirements of a framework which can provide solutions to the predefined problem and finally present a conceptual outline towards the solution of that problem.

In Chapter 5, we proceed with the experimental part of our study. We explain the purpose, the methodology adopted along with the decisions behind the experimental design choices and provide possible threats to it's validity.

Additionally, in Chapter 6, we perform statistical analysis in order to identify the significance of our experimental findings, we interpret the outcome of that analysis and discuss its implications.

Finally in Chapter 7, we submit our conclusions by providing an answer to the research questions, we realize the limitations of this current study and examine some possible future work areas.

Chapter 2

Related Work

In this section, we will discuss several subjects that are related to the research at hand. First, in Section 2.1, we will discuss the fields of software evolution and maintenance which can be found under the context of software engineering. Next, in Section 2.2, we will discuss what research has been conducted on the field of user involvement in software development. We will also explore in Section 2.3 the world of gamification in general and we will include some research done concerning gamification in the field of feedback acquisition.

2.1 Software Evolution and Maintenance

Existing research can be found for both software evolution and software maintenance and a related review will be presented in this subsection. Software evolution is an important component of software engineering [22] and it involved many different evolution processes. Software engineering continues throughout the lifetime of a software system and does not stop upon a system deployment. There's always the need for change in parts of a software system. Modifications made in order to correct errors that are found in operation, adaptation of the system for adjustments to its hardware and software platform and improvement in its performance or requirements and other non-functional attributes. Moreover, systems have to evolve in a “systems-rich” environment, which often increases the difficulty of the evolution process. Furthermore, understanding and analyzing the influence of a proposed modification on the system itself, someone may also have to assess how this modification may influence other systems in the operational environment [22]. Therefore, software engineering could be perceived as a spiral process with requirements, design, implementation, and testing going on throughout the lifetime of the system (*Figure 2.1*).

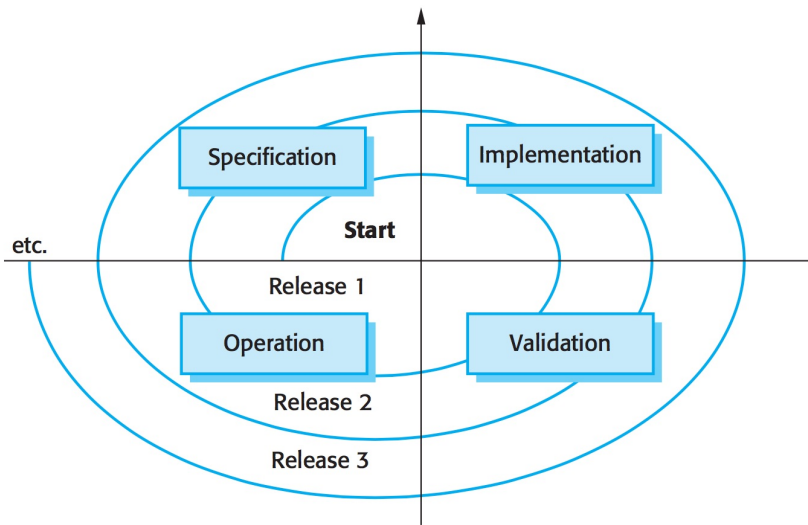


Figure 2.1: Spiral model of development and evolution.

This model of software evolution indicates that a particular corporation is pledged with the initial software development along with the evolution of the software. Most packaged software applications are being developed by adopting this approach. For a tailor-made software, a different approach is frequently used. A software company develops software for a customer organization and the customer's own development personnel then is responsible for that software. Alternatively, the software customer might hire a separate a different company which will be responsible for system's support and it's evolution [22].

Software evolution processes differ depending on the development processes used in an organization, the type of software being maintained, and the skills of the people consisting the development team. In some organizations, evolution may be an unofficial process where revision requests mainly originate from conversations between developers and the system users. On the other hand, in some companies, evolution is a formal procedure with organized documentation composed at very stage stage of the process. System modification requests play the most significant role in system evolution in most software development organizations. Modification proposals may originate from requests for new requirements, existing requirements that have not yet been implemented in the released system, bug reports from system stakeholders, and fresh ideas for software improvement from the development team involved into the project. The processes of change identification and system evolution continue throughout the lifetime of a system and are recurrent [22] (Figure 2.2).

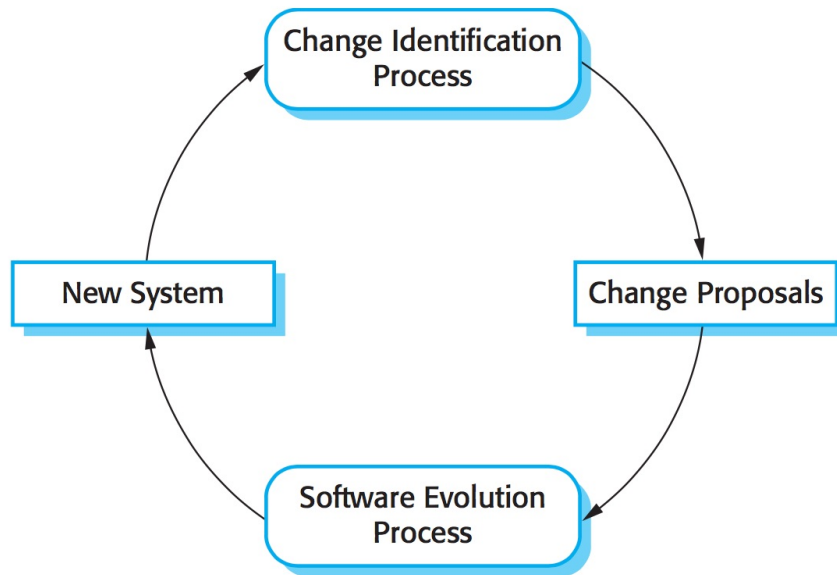


Figure 2.2: Identification and evolution processes.

According to Sommerville’s “*Software Engineering*” book [22], software maintenance is defined as the general process of changing a system after it has been deployed. This term is usually referring to custom software applications in which separate development teams are involved before and after the software’s release. Modifications made to the software may be simple, changes to address coding errors, more extensive to address design errors, or significant enhancements to accommodate new requirements or address specification errors. These changes are implemented by modifying current system components and by adding new ones to the system where its necessary. According to [22] there are three different forms of software maintenance:

1. *Fault repairs*: Errors in code which are usually relatively cheap to correct; error in the design which are more expensive as they may involve rewriting several code areas. And requirements errors which may require extensive system redesign, thus they are the most expensive to repair.
2. *Environmental adaptation*: This form of software maintenance is appropriate when some aspect of the systems environment such as the platform operating system, the hardware, or other support software is modified. The application system must be altered in order to adapt to these environmental changes.

3. *Functionality addition*: This form of software maintenance is crucial when the system requirements alter in correlation with business or organizational changes. The scale of these modifications needed to the software is usually much greater than for other forms of software maintenance.

These forms of software maintenance are universally accepted but sometimes named differently. “Corrective maintenance” is generally used to indicate to maintenance for fault repairs. Nevertheless, “adaptive maintenance” sometimes involves adaptation to a new environment and sometimes indicates adjustment of the software to new requirements. “Perfective maintenance” refers to the improvement of the software by implementing new requirements; but also in other cases it indicates the improvement of a system’s structure and performance while preserving the key functionality of the system [22].

The process of system evolution involves understanding of the program that has to be modified and the implementation of these modifications. Nevertheless, applying changes can be a challenging process in many systems, especially in older legacy ones, due the difficulty of understanding such systems. These systems may have been optimized for space utilization or performance at the expense of understandability, or the initial system structure may have been depraved by continuous changes [22]. To maintain more easily legacy software systems, reengineering could be adopted in order to improve their structure and coherence. Reengineering may involve several actions such as translating programs to a modernized programming language, refactoring the system’s architecture, updating and modifying the values and structure of the systems data and redocumenting the system. The functionality of the software system is not changed and, normally, critical changes are avoided to be made to the system architecture. The main issue with software reengineering is that there are practical limitations about up to which extend a system can be improved. “It is not possible, for example, to convert a system written using a functional approach to an object-oriented system” [22]. Major architectural changes or radical reorganizing of the system data management can be very expensive since those processes cannot be executed automatically. Although maintainability can be improved with reengineering processes, maintaining the reengineered system will probably not be as easy as maintaining a new system which is developed using modern software engineering methods such as refactoring.

“Refactoring is the process of making improvements to a program to slow down degradation through change” [17]. Refactoring involves program modifications in order to reduce its complexity, or to make it easier to understand and improve its structure. It is considered to be suitable to object-oriented

development but the its principles can be adapted to any development concept. Refactoring can be thought of as preventative maintenance that reduces the problems of future change since it involves changes that focus on the improvement of the system. Although the process of reengineering and refactoring are both intended to improve understandability and change of a software system, they are not the same thing [22]. Reengineering takes place after the maintenance of a system. Adoption of automated tools will allow the processing and reengineering of an existing system in order to create a new system that is more maintainable. Refactoring can be a continuous process of improvement throughout the software development and software evolution processes. It is intended to avoid the structure and code degradation which will eventually increase the difficulties and costs of maintaining a system. Refactoring can be considered as an inherent fraction of agile methods such as extreme programming, since such methods are mainly based around change. Program quality is therefore liable to degrade fast, thus agile developers usually adopt a refactoring approach for their programs to avoid this degradation [22]. Moreover, the use of refactoring with emphasis on regression testing in agile methods lowers the risk of introducing new faulty behavior of the system. Nevertheless, refactoring is not dependent on other “agile activities” and could be used along with any software development technique. Fowler [37] implies that there are stereotyped situations (he calls them “bad smells”) in which the code areas of a program can be upgraded. Examples of “bad smells” that could be improved with the use of refactoring are the following [37]:

1. *Duplicate code*: The same of very similar code may be included at different places in a program. This can be removed and implemented as a single method or function that is called as required.
2. *Long methods*: If a method is too long, it should be redesigned as a number of shorter methods.
3. *Switch statements*: These often involve duplication, where the switch depends on the type of some value. The switch statements may be scattered around a program. In object-oriented languages, polymorphism can often be used to achieve the same thing.
4. *Data clumping*: Data clumps occur when the same group of data items (fields in classes, parameters in methods) reoccur in several places in a program. These can often be replaced with an object encapsulating all of the data.
5. *Speculative generality*: This occurs when developers include generality in a program in case it is required in future. This can often simply be removed.

Fowler, in his book and website [37], considers also some basic refactoring transformations which could be used individually or simultaneously to address the “bad smells”. Examples of these transformations include Extract method, where duplication is removed and a new method is created; Consolidate conditional expression, where replace a sequence of tests is replaced with a single test; and Pull up method, where subclasses with a single method in a super class, replace many similar methods. Integrated development environments, such as Eclipse, include refactoring functionality. A fact which makes it easier to identify parts of a program that require modifications in order to apply the refactoring. Refactoring, executed during development, is an effective way to reduce the long-term maintenance costs of a software system. Nevertheless, maintenance of a system whose structure has been significantly degraded, then it may be practically impossible to refactor the code alone [22].

Takeaway: By examining the principles and functionality of software evolution and maintenance in the previous section, we can understand the importance of the role of such processes in the field of software engineering in general. Efficient and successful software maintenance and evolution processes could lead to better quality software applications, which could meet the end-user’s requirements and preferences. We would like to in this present study to examine if there is room for improvement and which possible strategy could be adopted towards achieving more effective software maintenance and evolution practices. In order for this to be more feasible, user involvement should be taken into account as an effect mechanism to software maintenance and evolution processes.

2.2 User Involvement in Software Evolution

User feedback contains important information for developers, helps in the improvement of software quality and to the identification of features missing from a current software version. Its aim is to maximize system’s usability and usefulness by identifying users expectations and requirements. Over the last three decades research concerning user involvement in software engineering has been conducted; and it seems that user involvement is playing a more and more important role in the system developing cycle [15]. User feedback is a primary source for acquiring relevant information needed for planning software evolution and adaptation [6].

Stakeholder involvement in software maintenance and evolution processes is an important factor to increase the success of a software application. A study performed by Kujala [43] shows that there is a significant equivalence

between user involvement and the success of information system development as well as user satisfaction. An early study by Baroudi [24] has shown that a greater user involvement leads to a higher software application usage and admission.

The success of any development project relies on overall user orientation, which be achieved by meaningfully involving users in systems development process. User involvement has changed significantly over the last thirty years. For instance, software users changed from programmers or trained technical personnel to basically any person [27], provoking a significant shift in developers attitude towards them [7]. With the use of application distribution platforms and the constant increase of use of mobile devices, neither the users of software nor its context of use are known before the software's release. As a consequence, these changes widen the distance between end users and software developers [28], while more focus on users would actually be vital to satisfy their increasing needs [46]. Therefore, post-release user feedback such as bug reports and feature requests become increasingly crucial to software developers [5]. Especially in the maintenance and evolution phases, in case the users come across bugs while they use a system, they should inform the system's developers immediately for evaluation.

Another aspect is the fact that users may have new requirements about the system; they provide the initial information for the system developers to help them in defining the new problems [44]. User involvement can promote the clear definition, although sometimes the users might provide the system developers with inaccurate information. Thus, better understanding of the problems occurring and the feedback to be provided by the users, can be reached only by promoting effective communication between them and the development teams during all processes of the software life cycle [40].

Dodd & Carr [16] consider that the end user is the first member of a team organized to define problems. Users may participate in prototyping, data gathering and data flow diagrams reviewing. System development methodologies could benefit in several ways from adoption of user involvement:

- User involvement *helps system developers identify* the current problems that might be neglected because lack of environment understanding [16].
- As stated by Dodd & Carr [16], user involvement *can avoid the conflict between users and data services*. Temporarily ignoring the conflict between the continuity of system development cycle and the limitation of the budget and time, user should be involved throughout the system

development activities. In this way system developers can communicate with them at any time for meeting latest needs and improve the data service.

- *“Jointly involving users and systems professionals helps create an understanding of why trade-offs are mad”* [16]. If the systems professionals don't communicate with the users, it is very possible for users to complain that the delivered system doesn't fit with their requirements although they believe that product is appropriate. However if users are involved into the whole development process and are encouraged to communicate with developers, they can reveal the existing problem situation better. Meanwhile, during development process, users can give immediate feedback when they find any mismatch between system design and expected requirements. In these ways, the trade-offs can be more reliable and reasonable.
- User involvement *improves the overall computer literacy of the company and provides better understanding of computer-based technology and the systems development process for users.* “This creates a more knowledgeable user community, one who is better able to request and use such technology” [16]). User involvement is a mutual benefit process. When users involve in the system development cycle, besides the benefits for developers getting better understanding of current system, users also can gain the opportunity to learning and studying the new system, because it is a part of its creation. All of these will contribute to further system implementation and use.
- User involvement *provides insights into how individual works impact the departments.* Users and data services staff become more attuned to a systems perspective of the whole company that leads to the department integration within the company and work becomes more efficient. Dodd & Carr [16] pointed out the close working relationship of data services and users create a consensus of purpose that eliminates the “we-versus-they” point of view. With these distinguished advantages, user involvement has been realized as one effective technique for effective methodology implementation. Consequently, the technique is adopted by several system development methodologies.

Takeaway: Adoption of user involvement provides several benefits for system development methodologies as mentioned on the section above, thus we would like to investigate further how involving the users could affect the processes of software maintenance and evolution. According to Zhiwei [44], user involvement is playing a more and more significant role in the system developing cycle. Thus, certain measures should be taken in order to approach and stimulate the interest of the users in assessing software products. A

possible way intrigue and motivate end users to provide their feedback and get involved more in current software development practices could be the use of gamification.

2.3 Gamification

According to Ali & Dalpiaz, users generally lack motivation and interest in getting involved into providing their comments and feedback, especially in a such a way that would actually make a difference for software developers. In their paper about “Gamified Culture-aware Feedback” they state that gamification could be a technique to maximize users motivation and change their reaction to feedback requests [35]. Zichermann & Cunningham on their book “Gamification by Design”, they define the term gamification as follows:

“The process of game-thinking and game mechanics to engage users and solve problems”

This flexible and powerful framework for understanding gamification, it can easily be applied to any problem that can be solved by influencing human motivation and behavior. Gamification brings together all the various aspects that have been advanced in games for non gaming contexts. “Games are generally good motivators” [21]. Games have become one of the most powerful forces by focusing on three central components pleasure, rewards, and time. Uniquely, games are able to get people to take actions that they dont always know they want to take, without the use of force, in a predictable way [21]. The success of games relies in an idea called *flow*. The perception of flow is derived from the work of Mihaly Csikszentmihalyi, a psychology professor who is renown for his studies about creativity and happiness. Achieving flow, indicates a player’s state between anxiety and boredom, meeting his own motivational level in that experience (*Figure 2.3*) [21].

Although the gamification concept might be motivating for people, the same game rules might not apply for different groups of people. According to Bartle, player types are a way of classifying individuals according to specific psychological aspects of their personality and how they prefer to act in a virtual environment (*Figure 2.4*) [39].

This classification is based on the compilation and observations of the results of a forum discussion (that lead later to The Bartle Test - which, despite bearing his name, was not created by Bartle) between players about what they thought was fun in a game and what they thought others found fun about the same game [21].

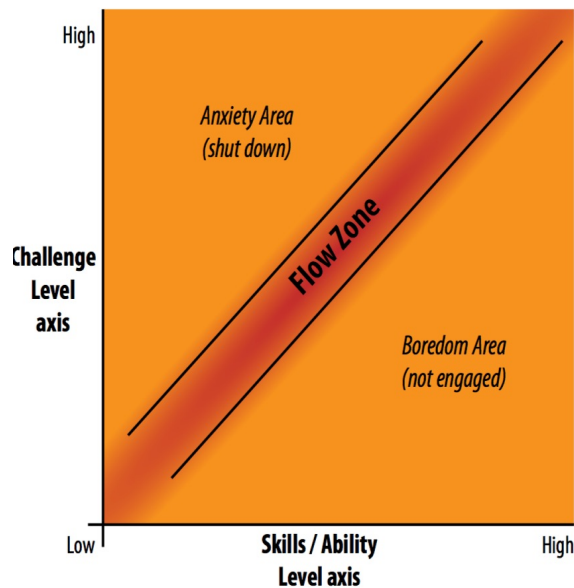


Figure 2.3: The state of flow is achieved when a player is placed between anxiety and boredom over a period of time.

From the summary and observations of the discussion, Bartle denotes that players could be split into four types; where players could also have same different levels for each of these four types (e.g. a player can be 70% killer and 30% achiever), providing psychological portraits of individuals which inhabit a virtual world [39] [3] [30]:

- *Killers* like to provoke and cause drama and/or impose them over other players in the scope provided by the virtual world. Trolls, hackers, cheaters, and attention farmers belong in this category, along with the most ferocious and skillful opponents [39].
- *Achievers* are competitive and enjoy beating difficult challenges whether they are set by the game or by themselves. The more challenging the goal, the most rewarded they tend to feel [39].
- *Explorers* like to explore the world - not just its geography but also the finer details of the game mechanics. These players may end up knowing how the game works and behave better than the game creators themselves. They know all the mechanics, short-cuts, tricks, and glitches that there are to know in the game and thrive on discovering more [39].
- *Socializers* are often more interested in having relations with the other players than playing the game itself. They help to spread knowledge

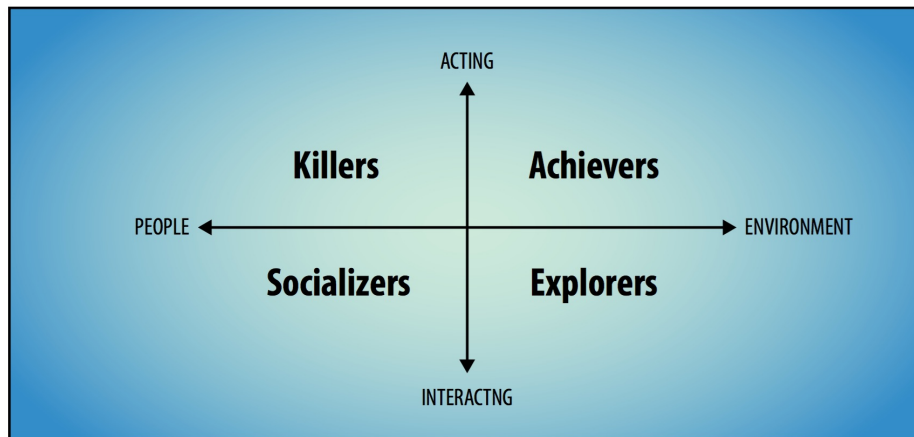


Figure 2.4: Bartles player types.

and a human feel, and are often involved in the community aspect of a game [39].

Bartle calls it a “bandwagon”. Meaning that not every game design will be effective for all types of players, thus many different design approaches should be used in order for gamification to be effective [39]. For instance, a shoe selling website that provides points purchase of a pair of shoes. By acquiring a specific amount of points, the buyer could gain access to a certain pair of shoes he or she could only buy due to the points already acquired. This concept is probably viable and could effectively work for achievers type. On the other hand, an individual who belongs to the explorer type, is visiting the entire web page, exploring every aspect of it. Points are awarded to him or her for this specific action. These points would be worthless - rather, one should reward the action of exploring by providing a way to continue to interact and enjoy the web page. Under the same concept, a leader board will not necessarily affect the actions of a socializer type. This type of player would probably not be so interested in rating, getting to chat with and meeting new people might be much more interesting.

Takeaway: Understanding the context of the main principles of gamification will allow us to use the correct and appropriate incentives to engage the users. Adoption of gamification could provides several benefits for software application development methodologies [24], thus we would like to investigate how user involvement could be affected by providing them with the appropriate stimulus in order to provide their feedback. Thus, certain measures should taken in order to approach and stimulate the interest of the users in assessing software products. Under this context, we can assume that by using the right intrinsic or extrinsic incentives depending on the group

of people we are addressing; gamification could be used to allow software versatility also for feedback acquisition purposes[35].

Chapter 3

Approach and Design

In this chapter, we present the approach and the design of our framework. Firstly, we are going to examine the problem domain and identify the scope of the problem to be solved (*Section 3.1*). Secondly, we are going to describe our conceptual framework towards finding a solution to the aforementioned problem (*Section 3.2*). Lastly, we are going to provide our approach for delivering a solution to that problem by providing the design and architecture for that conceptual solution (*Section 3.3*).

3.1 Problem Domain

In the introduction of this document, the problem that drives this study has been outlined. As described in Chapter 2 also, feedback acquisition is a very important part of the software evolution and maintenance processes. That explains why in our research we tried to investigate and construct an instrument which could increase the effectiveness of current feedback acquisition mechanisms and also provide some incentives in order to engage the end users.

Pagano and Bruegge [15] on their work about user involvement in software evolution are providing us with the following results:

1. *There are four main user feedback artifacts:* Error reports, feature requests, feedback on existing features, ratings.
2. *User feedback is scattered:* email, application distribution platforms, integrated feedback mechanisms etc.
3. *Users intentionally select feedback channel:* The more critical the feedback, the more public the channel.
4. *Users are not systematically involved:* No agreed practice how to provide nor how to gather user feedback.

Remaining informed of the users' opinion on the role of software systems in order to meet their constantly changing needs is crucial. Current practices do not involve systematically the user in feedback acquisition process, neither provide the appropriate incentives that motivate and engage the public for providing their feedback. Solutions are going to be essential to meet the needs of businesses, where change is constant and urgent [31]. Users need stimuli that make them eager to contribute to the process.

3.2 Conceptual Framework Requirements

By examining current studies, it is clear that there is still a big gap between development teams and users, thus it is still quite difficult to gather, process and understand user's comments concerning a software product. The overall objective of this study is to create and validate a feedback acquisition framework which will be embedded in already existing applications and will engage and motivate the users by exploiting the advantages of gamification elements. The following table contains a list of key functional requirements. These functional requirements were prioritized with the MoSCoW approach to determine their importance [32] (*Figure 3.1*). Along with the functional requirements, we have identified the non-functional requirements our tool should meet (*Figure 3.2*).

We would like our framework to be able to embed a gamified feedback acquisition mechanism to already existing applications and redirect feedback gathered to feedback report/resolution platforms (*Table 3.1, Requirement Category 1*). Thus, the tool should be able to refactor the original code of a software application and apply our feedback acquisition mechanism (*Table 3.1, Requirement Category 2*). Our feedback acquisition mechanism will enable certain feedback types gathering (e.g. comments, bug reports, feature requests, etc.) for the application as a whole but also for specific elements of the application (*Table 3.1, Requirement Category 3*). Moreover, it will redirect the feedback gathered to a feedback report platform (e.g. forums, bug resolution platform, mailing lists, etc.) (*Table 3.1, Requirement Category 4*). Last but not least, the tool will use a certain point rewarding system as part of the context of gamification in order to motivate and engage the users (*Table 3.1, Requirement Category 5*).

This approach can be expressed as an effort towards addressing the aforementioned issues *Section 3.1*, identified by Pagano and Bruegge [15] on their research about current feedback acquisition practices. In our framework we will include the main user feedback artifacts (error reports, feature requests, comments, ratings). We will accumulate all feedback gathered and direct it into one feedback report/resolution platform. And provide suitable user

incentives in order to engage and motivate the end users, while ensuring the usability of our tool.

3.3 Conceptual Solution Outline

In order to address the effectiveness of feedback acquisition mechanisms we started by investigating how current practices are requesting user feedback and how users are being involved into the process. Our approach involved a hands - on experience with modern general use software products with focus on their feedback acquisition mechanisms. Such software products were

Category	Description	Priority
1. Existing IDE extension	• Tool developed on top of the functionality of an IDE	High
	• Tool developed on top of multiple IDEs	Medium
2. Source code refactoring	• Refactor application's original source code	High
	• Inject feedback functionality to the application	High
3. Feedback Acquisition	• Embed feedback acquisition functionality on the application	High
	• Enabling feedback for certain application elements and the application as a whole	High
	• Enable multiple feedback acquisition interfaces	Low
4. Feedback Reporting	• Embed feedback reporting functionality on the application	High
	• Decide feedback platforms to report the feedback acquired	Medium
	• Enable feedback analysis mechanism	Low
5. Gamification elements adaptation	• Enhance the process with gamification elements	High
	• Provide incentives to the users	High
	• Adapt gamification elements according to user types	Medium
	• Enable multiple gamification elements	Low

Figure 3.1: Overall Framework Functional Requirements.

modern productivity platforms such Microsoft Office, Open Office, Libre Office and Apple’s iWork bundle. We examined how these software products, which are meant for use by non expert end users, are requesting feedback from their users and how they try involve them more in this process. We identified that the most commonly used practices are to use forums and discussion groups in order to gather feature requests for their products, bug reporting and resolution platforms for bug reports and fixes, along with mailing lists to provide support to their users. Moreover, other practices are involving focus groups and and interviews with users in order to identify their preferences and extract their remarks or criticism. Our investigation and the aforementioned current feedback acquisition practices, is supported by the findings of Pagano & Bruegge [15] on their work about user involvement in software evolution.

In order to start with the design of our approach we had first to investigate further which current feedback acquisition techniques are the most commonly adopted. There are two techniques currently being by experts; explicit and implicit feedback. According to G. Jawaheer on his work about these two different types of feedback [20], explicit feedback is considered as a technique which involved rating scales, provides users with a mechanism to decidedly express their interests in items. Explicit feedback is in general more accurate than implicit feedback in the representation of user’s interests (this is of course dependent on the application and the domain). Moreover, explicit feedback can be either positive or negative, whereas implicit feedback is only positive. Furthermore, explicit feedback concentrates on either

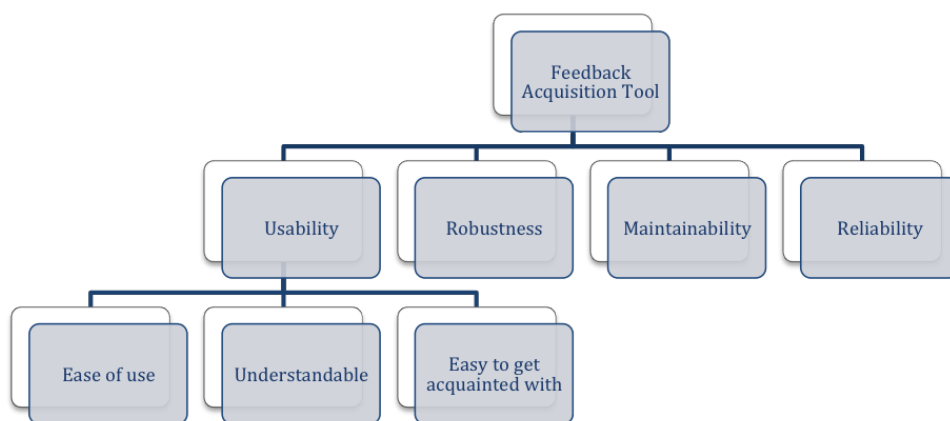


Figure 3.2: Overall Framework Non-Functional Requirements.

side of the rating scale, as users tend to express their preferences if they feel strongly for or against an object.

On the other hand, implicit feedback is created by the software application itself, through inferences it makes about the user's behavior. The constitution of implicit feedback depends on the application domain: Usually, it will be one or multiple measurable and observable parameters that are derived from the user's interactions with the application. Most of the research in software applications has focused on using one or the other type of feedback; only few have combined these two heterogeneous feedbacks [47].

We decided to follow an explicit feedback design for our framework. This decision is based on the features of the explicit feedback technique which would ensure our tool can be used in different types of applications without much customization or modification to a developer's needs. We conceived the following explicit feedback scenario in which our tool will be able to reciprocate (*Figure 3.3*):

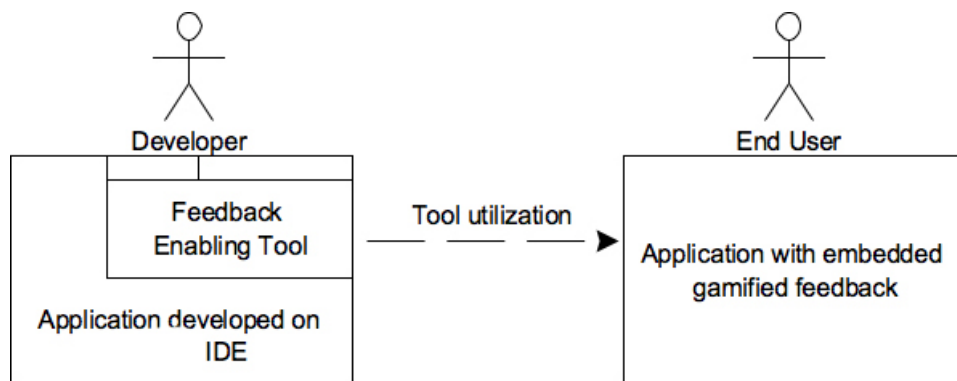


Figure 3.3: Explicit Feedback Scenario.

According to *Figure 3.3* while using the tool:

1. *The Developer will be able to use the feedback enabling tool on the IDE.*
2. *The feedback enabling tool will refactor the application's code.*
3. *The end user will be able to provide his/her feedback within the application.*

The main idea behind this conception is to allow the developer to enable or disable the feedback acquisition mechanism according to his liking on the application he is developing *Figure 3.3*. The developer will be able to enable

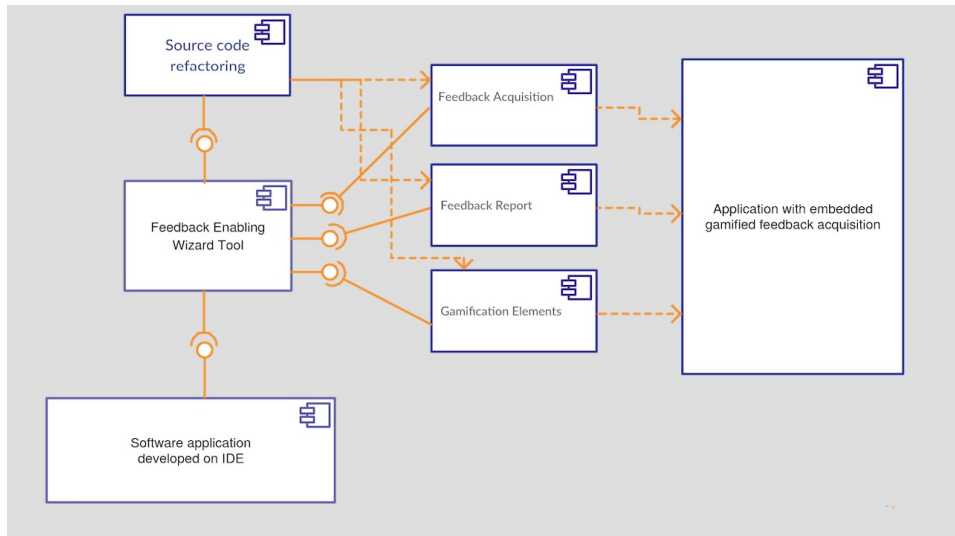


Figure 3.4: Framework Component Diagram.

or disable the feedback mechanism during development time and adjust its capabilities according to his preferences. This way he will be able to produce an end product with a feedback mechanism already embedded, without the need to create from scratch a different - external feedback mechanism. Furthermore, the developer will be able to decide what kind of feedback he wants to retrieve from the users and where this feedback should be stored and processed. Last but not least he will have the capability to motivate and engage the application's end users by enabling a rewarding system which is based on gamification practices.

In *Figure 3.4* and *Figure 3.5* we provide the component and activity diagrams of our tool. First of all a wizard will be used in order to acquire the developer's preferences. The developer will be able to choose the application's code to be affected, the feedback types to be acquired, the feedback storing platform to redirect the feedback gathered by the end users and the gamification elements to be used. According to the developer's preferences the tool refactors the already existing code of the application, enables the acquisition of the feedback types defined by the developer and directs the storing of the feedback gathered into the target platform of the developer's choice. The final output of our tool will be the same application previously implemented by the developer enhanced with gamified feedback acquisition.

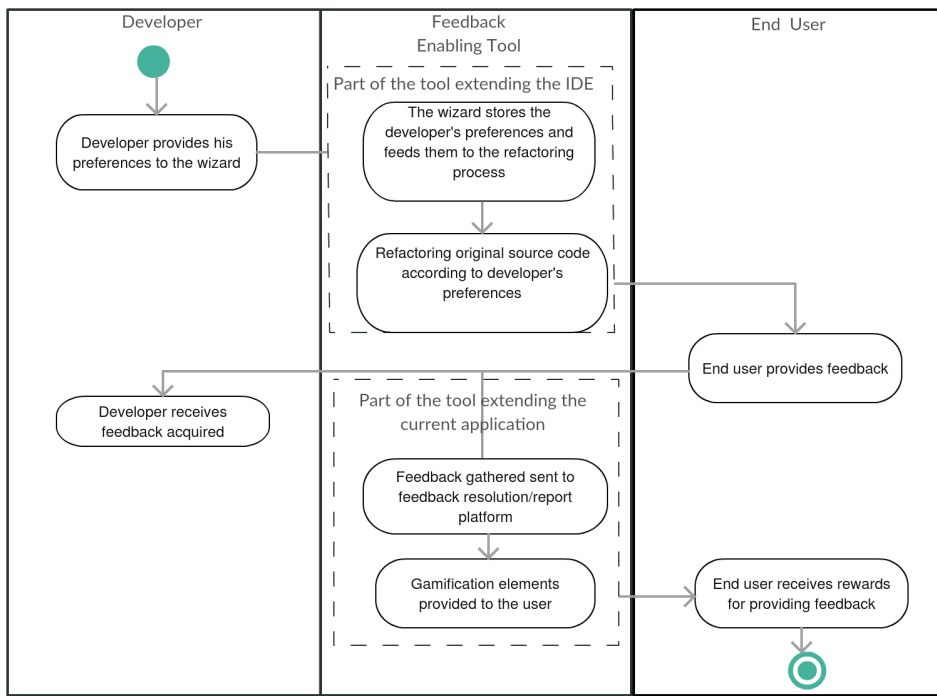


Figure 3.5: Framework Activity Diagram.

Chapter 4

Framework Implementation

In this chapter we present the implementation process and dive into the technical details which will clarify the functionality and purpose of our tool, which we are going to call “*Gamified Feedback Enabling Tool*”. Firstly, in section 4.1.1 we report on what kind of software and hardware was used for our framework implementation. Further, in section 4.1.2 we describe the software specific setup and follow up with framework architecture in section 4.1.3.

4.1 Generic Implementation Details

As previously mentioned in Chapter 3, our main goal is to develop a tool that could be applied to an Integrated Development Environment(IDE) and which can be used by developers in order to enable feedback acquisition for their applications in combination with gamification elements. The IDE to be used is Eclipse version Kepler and *Java* the programming language we used for development . Development and testing was executed on a 2.4 GHz Inter Core i5 processor Apple Macbook Pro with 8 GB 1600MHz DDR3 of RAM.

4.2 Framework Design

For the purposes of the development of our framework we decided to take a look at already existing open source software applications. This choice is based on the opportunity of using the source code of a software application under the open source license agreement without the need of acquiring the rights of proprietary software application. Moreover, open source software application involve a vast number of developers from all over the world who contribute into the evolution and creation of such applications and that way our framework could contribute to the open source community and to the effective development and maintenance of such software projects. In order

to identify which software applications are the most accepted and used by end users we did investigate the current download rates on platforms such as *sourceforge.net* and *github.com*. We determined the most popular applications to be general productivity applications such as *Open Office*, *Libre Office* and applications such as *JEdit*, *OpenCV* and *OpenProj*. These software applications also scored amongst the highest rated according to the users in these web-based repositories. By determining the applications that are most downloaded by end users we had to determine the programming language and the development environment these applications are being developed at. We identified that the most commonly programming language currently used by developers around the world is *Java* and that most of the developers are using *Eclipse* and *Netbeans* IDEs (integrated development environment). These aforementioned attributes of open source software applications determined which programming language our framework will be developed at and along with which already existing application our tool will be tested with. Thus, we decided that the programming language to be used is *Java*, the development will be executed in the Eclipse IDE and the application on which our tool be tested at is *JEdit*. Our implementation choices will be thoroughly explained in the paragraphs to follow.

Firstly, we created a plug in for the Eclipse IDE¹ and our tool's functionality is based on the Eclipse Plug-in Development Environment (PDE)². The Eclipse development environment consists of several Eclipse components. A component in Eclipse can be referred to as a plug-in. The Eclipse development environment allows a developer to extend the Eclipse IDE with supplementary functionalities with the use of plug-ins.

Eclipse applications use a runtime based on a specification termed OSGi³. A software component in OSGi is referred as a bundle which is always a plug-in. The Eclipse development environment forms the basis of one of the most successful Java IDEs and therefore is very stable and broadly used. It uses native user interface components which are fast and reliable. It has a strong modular approach based on the industry standard module system for Java (OSGi) that allows developers to design component based systems.

The Eclipse IDE version 2.0 started as a modular IDE application. In 2004 Eclipse version 3.0 was released. Eclipse 3.0 supported reusing components of the Eclipse platform to build stand-alone applications based on the same technology as the Eclipse IDE.

At this point the term Eclipse RCP was created. Eclipse RCP is short for Eclipse Rich Client Platform⁴ and indicates that the Eclipse platform is

¹www.eclipse.org/ide/

²www.eclipse.org/pde/

³www.osgi.org/

⁴wiki.eclipse.org/index.php/Rich_Client_Platform

used as a basis to create feature-rich stand-alone applications.

The release of Eclipse in version 4.x simplified and unified the Eclipse programming model which is now based on state-of-the-art technologies, like dependency injection and declarative styling via CSS files.

Eclipse RCP applications benefit from the existing user interface and the internal framework, and can reuse existing plug-ins and features.

The Eclipse IDE is basically an Eclipse RCP application to support development activities. Even core functionality of the Eclipse IDE is provided via a plug-in, for example the Java development or the C development tools are contributed as a set of plug-ins. Only if these plug-ins are present the Java or C development capabilities are available.

According to this description about RCP capabilities in the Eclipse environment, we decided to take advantage of all these features and create our own extension of the Eclipse editor by developing our feedback enabling tool as an Eclipse plug-in. The plug-in consists of two parts, a wizard which is responsible for acquiring the developer's options concerning on the functionality the tool will have. Thus, the wizard is the first thing a developer will encounter while using our tool.

The wizard consists of a series of questions which will give a certain number of choices to the developer and will define the functionality of the application of our tool. Firstly, a developer will have to decide which Java class of his application will be affected by the refactoring process and this class must be responsible for the design and implementation of the user interface of the application.

Secondly, the developer can decide about the position of a *Feedback* button to be placed in his application's already existing user interface. This button will be responsible for initiating the feedback acquisition process. Moreover, the developer will be able to decide in which platform the acquired feedback will be stored and also the reward system to be used in order to provide a set of incentives for the users.

As soon as the developer has provided his preferences to the wizard, then our tool is able to start the refactoring of the application's already existing code, define the placement of the feedback button, redirect the feedback gathered to the platform chosen and last but not least enhance the feedback experience with gamification elements which will serve as user incentives.

During the refactoring process our tool searches and identifies all the graphical user interface elements which are developed under the Java Swing⁵

⁵[en.wikipedia.org/wiki/Swing_\(Java\)](https://en.wikipedia.org/wiki/Swing_(Java))

framework. Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes⁶(JFC) an API for providing a graphical user interface (GUI) for Java programs. Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit⁷(AWT). Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

The first thing to be done by our tool is identify the Java project on the Package Explorer of Eclipse, identify the *source* folder and define the package which includes the current project's source files. As soon as the package identification has been completed, the tool creates an Abstract Syntax Tree⁸ which serves as a tree representation of the abstract syntactic structure of source code written in any programming language (e.g Java). Each node of the tree denotes a construct occurring in the source code (*Algorithm 1*).

This is possible by using the ASTParser⁹ class in Eclipse which is responsible for creating such ASTs for the Java programming language along with the use of the *Visitor* design pattern which allows for one or more operations to be applied to a set of objects at runtime while decoupling the operations from the object structure *Figure 4.1*.

Algorithm 1: AST Creation

```

ASTParser declaration;
Workspace identification;
JNATURE projects identification;
if project found then
    while we have not reached the end of the source folder do
        execute method identification;
        execute variable identification;
    else
        define Compilation Unit;
        parse AST's created branches by method and variable analysis;
        create AST;

```

⁶en.wikipedia.org/wiki/Java_Foundation_Classes

⁷docs.oracle.com/javase/8/docs/technotes/guides/awt/index.html

⁸en.wikipedia.org/wiki/Abstract_syntax_tree

⁹inst.eecs.berkeley.edu/~cs164/sp14/javadoc/ASTParser.html

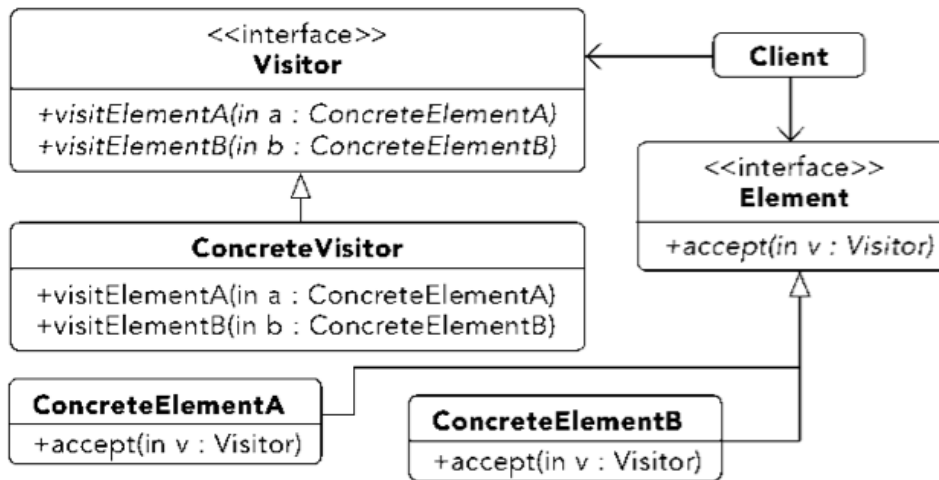


Figure 4.1: Visitor Design Pattern.

We decided to use this specific design pattern along with its implementation in Eclipse as an extension of AST framework. Thus we used the `ASTVisitor`¹⁰ class which allows for a preorder traversal of the abstract syntax tree when previsiting each abstract syntax tree node and does a postorder traversal of the abstract syntax tree when it performs a postorder traversal of each abstract syntax tree node. That way we can affect each node of the tree on runtime and perform changes on the nodes that the pattern visits. This process will serve the purpose of identifying the already existing source code's methods and variables and apply the changes required for our tool to perform (*Algorithm 2 and Algorithm 3*).

Algorithm 2: Method Identification - Visitor Pattern

```

method arraylist declaration;
if methods found then
    while we have not iterated through all the project's classes do
        visit class;
        identify method;
        create visited node;
    else
        save methods identified to arraylist;
    return method arraylist;
  
```

¹⁰docs.basex.org/javadoc/org/basex/query/util/ASTVisitor.html

As soon as the AST is created and the ASTVisitor is able to traverse through its nodes, the tool is searching for all the Java Swing components inside the Java class defined by the developer on the wizard pane and identifies the name of the top component which in Java Swing is a JFrame¹¹. In this specific JFrame we are able to identify all of its sub components which the user interface of the application consists of. Moreover, all the variable names, types and positions values of these components are being identified in order to add an additional feedback functionality to them. That additional feedback functionality will serve as a user interface element specific feedback feature, which will allow the end user to provide his or her feedback for that specific user interface element.

By identifying the positions of the user interface elements we are able to define the position of a *Feedback Button* to be added on the user interface of the developer's application. That *Feedback Button* will be responsible for triggering the feedback acquisition process on the application for a certain user interface element, the application functionality in general or both. Furthermore, it will redirect the user to a user friendly environment to provide his or her remarks for the application while providing some incentives under the concept of gamification. The functionality of this *Feedback Button* is added on the project's current source code by injecting a new method on the current user interface class. This method serves as an action listener and when the button is pressed it enables the feedback functionality on the application and all its components.

In order to complete the refactoring process all the aforementioned changes have to be applied to the original source code of the application. This was made

¹¹docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html

Algorithm 3: Variable Identification - Visitor Pattern

```
variable arraylist declaration;  
if variables found then  
    while we have not iterated through all the project's variable  
        declarations do  
            visit class;  
            visit methods identify variable; create visited node;  
else  
    save variables identified to arraylist;  
    return variable arraylist;
```

possible by using the `ASTRewrite`¹² class, which serves as an infrastructure for modifying code by describing changes to AST nodes. The AST rewriter collects descriptions of modifications to nodes and translates these descriptions into text edits that can then be applied to the original source.

The most significant point is that this process is done without actually modifying the original AST, which has the quality of allowing one to entertain several alternate sets of changes on the same AST (e.g., for calculating quick fix proposals). The rewrite infrastructure tries to generate minimal text changes, preserve existing comments and indentation, and follow code formatter settings. On this step we are defining for the `ASTRewrite` the insertion position of the code to be added to the original source code along with its parameters, variables and functionality. By providing these information to the `ASTRewrite`, we are able to apply all the modifications to the application's original source code and provide an end application with embedded feedback functionality (*Algorithm 4*).

Algorithm 4: AST Rewrite

```
ASTRewrite declaration;
Feedback Method definition;
{

    Functionality of the feedback button;

    Mouse cursor position identification;

    Feedback functionality added to user components;

}
Parse created AST of the application;
Define code area to inject the Feedback Method;
redefine AST with ASTRewrite;
return ASTRewrite output;
```

For testing purposes we designed and developed our own *Agenda* application which could serve as a simple scheduler and contact retriever. This application was developed under the Java Swing framework and was used for experimentation purposes for our tool. The *Agenda* application was designed and created according to the modern standards as far as user interface

¹²www.cct.lsu.edu/~rguidry/eclipse-doc36/org/eclipse/cdt/core/dom/rewrite/ASTRewrite.html

design is concerned in order to able to serve as an experimentation base for the development of our tool.

All the aforementioned processes concerning the wizard functionality, the user interface elements identification and the refactoring, have been performed and tested on this specific application first in order to determine first of all the feasibility and later on the performance of our tool. As soon as we established the key functionality for our tool, we worked on improving it's performance and functionality and decided to attempt an assessment on an original software application.

The software application to be chosen is JEdit¹³, which is a free software text editor available under the GNU General Public License version 2.0. It is written in Java, runs on any operating system with Java support, including BSD, Linux, Mac OS X and Windows and retains very high download rates and rating on *sourceforge.net*.

The choice of working with JEdit and test our tool's capabilities on it, is based on the fact that it is developed with the Java programming language and also under the GNU General Public License in order to have access to its source code. We decided to test our tool on an original software application in order to examine how smooth it would perform and define its strong aspects and it's weaknesses but also we wanted to perform a case study in order to identify the general users' opinion about our feedback mechanism embedded on a text editor such as JEdit. The case study analysis and it's findings will be explained more thoroughly in the following chapters.

4.3 Framework Architecture

As previously mentioned, the functionality and the processes executed by our tool are developed using the Java programming language. An overview of our system architecture can be seen in a class diagram, depicted in *Figure 4.2*.

As previously mentioned in order to deliver a tool which takes into account the developer's preferences, a wizard is appropriate to obtain such preferences. Thus the wizard is the first functionality given to our tool and *Feedback Enabler Wizard* is responsible for all the functionality that the wizard is providing according to the aforementioned description.

¹³www.jedit.org/

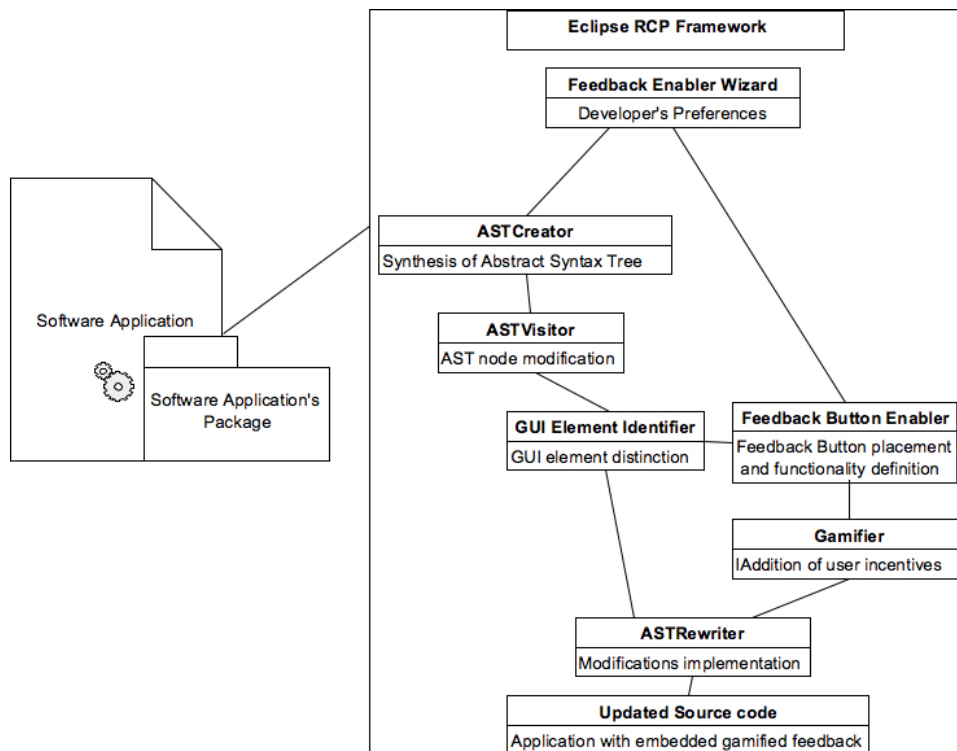


Figure 4.2: Class Diagram of our framework for enabling gamified feedback on an existing software application.

As soon as the developer's preferences have been acquired by the wizard, the *ASTCreator* class is responsible for analyzing the original source code of the application and creating the Abstract Syntax Tree in order to provide the structure of the whole project and provide the tree nodes needed for the processes to follow.

As long as the Abstract Syntax Tree is constructed, the *ASTVisitor* class is pledged with the task to attempt "visiting" each tree node in order to identify the components that the graphical user interface consists of. With the help of the *GUI Element Identifier* class, all the component names and types are being identified in order for the tool to be able to enable the feedback acquisition mechanism for each user interface element specifically but also adjust the Feedback Button position.

The *Feedback Button Enabler* class is gathering the information provided by the *ASTVisitor* and the *GUI Element Identifier* classes and is positioning a Feedback Button on the user interface accordingly (positioning done without allowing any overlaps or replacements of any already existing buttons or options in the user interface). A method (an action listener) is created

in order to provide functionality for this specific new button and also its response is defined when the user is interacting with it. The behavior on user interaction is specified to redirect the user to a feedback gathering platform when the user chooses to provide his or her feedback for a certain application element or the application as a whole. The *Gamifier* is applying the gamification elements which will serve as user incentives and will be visible to the user as soon as any kind of feedback is provided.

Last but not least, the *ASTRewriter* class is bound to gather all the code additions and modifications done previously by all the aforementioned steps and update the original application's source code. This way, the whole process is completely automated and the final output is a resulting application with embedded gamified feedback.

Chapter 5

Experiment

In order to evaluate our developed tool and validate its effectiveness and usability, an empirical study was conducted within the setting of a controlled experiment. According to Wohlin [13], a controlled experiment in software engineering is an empirical inquiry that manipulates one factor or variable of the studied setting. Based in randomization, different treatments are applied to or by different subjects, while keeping other variables constant, and measuring the effects on outcome variables. In human-oriented experiments, humans apply different treatments to objects, while in technology-oriented experiments, different technical treatments are applied to different objects. Experiments are employed when we want control over the situation and want to manipulate behavior directly, precisely and systematically. Also, experiments involve more than one treatment to compare the outcomes [13].

There is an increasing understanding in the software engineering community that empirical studies are required to improve processes, methods, and tools for software development and maintenance. Wohlin [13] is also discussing that computer scientists should experiment more while they are researching in the field of software engineering. This remark is justified by denoting that an empirical research can build a reliable knowledge base, lead to new and unexpected insights and help in the discovery of unexplored territories.

This chapter contains the scope of the experiment, the design which determined the participants to be included and the data to be acquired, along with the instrumentation to be used and the possible threats to validity.

5.1 Experiment Scope

Firstly, we determine the scope of the experiment by defining its goal and purpose. The purpose of conducting and executing this experiment is to

evaluate the usability of our tool. We would like to examine if the users were satisfied and enjoyed interacting with the tool by evaluating the tool’s ease of use, its coherence and accessibility. Moreover, we would like to examine the effectiveness of the use of gamification principles in the feedback acquisition process. Hence, we propose the following scope for our experimental research:

- *Evaluate the usability of our developed tool.*
- *Determine the effectiveness and influence of the application of gamification principles in the feedback acquisition process.*

5.2 Experiment Design

5.2.1 Design Choices

In order to evaluate the usability of our tool, we used the System Usability Scale (SUS)¹ test, which is a reliable tool for measuring the usability of a software application [25]. It consists of a 10 item questionnaire with five response options for respondents; ranging from *Strongly Disagree* to *Strongly Agree*. The SUS test was used to measure the usability of both our developed feedback acquisition mechanism and an already existing one. The already existing feedback acquisition instrument that was used in order to compare the usability results of the SUS tests was *Mantis Bug Tracker v.1.2.19*². As stated by its creators, “Mantis” is an open source issue tracker that provides a delicate balance between simplicity and power. “Once you start using it, you will never go back!”

The usability of our tool and the level of influence of gamification elements to the participants were evaluated in an experiment conducted in a university library setting. The participants who took part in the experiment, were asked to use JEdit and execute a series of activities which are described in Section 5.3, there was the possibility to report any issues the encountered and provide their feedback. At the end of this process they were asked to fill in the SUS questionnaire and provide any more remarks related to the feedback gathering tool they used.

The experiment was intended to compare the results obtained from a treatment group against a control group (*Figure 5.1*). In this experiment the treatment group could provide their feedback by using our tool which was embedded within JEdit’s environment and included gamification elements. While on the other hand, the control group could provide their feedback

¹www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

²www.mantisbt.org/

by using the *Mantis Bug Tracker* tool. The goal of the experiment was then to measure the SUS scores of both group’s participants and identify the quantity and quality of the bugs reported by both feedback acquisition instruments.



Figure 5.1: Experiment participant distinction.

5.2.2 Participant Selection

In Wohlin’s work is denoted that the selection of subjects must be representative for the population and ideally randomly chosen. Therefore, we decided to conduct the experiment in a university library environment and involve young people originating from different countries and from different fields of expertise.

Although it is assumed that a good sample size should consist of minimum 30 people to obtain statistical significant results [19], we had to rely on people’s desire to participate in such experiment. Both people’s desire in participating to the experiment and lessons learned from the pilot experiment led to an allocation of 10 people per group. The experiment used a balanced design, which means that the participants were divided into two groups with equal number of subjects. Balancing is desirable because it both simplifies and strengthens the statistical analysis of the data according to Wohlin [13]. Once equivalent groups were formed, both were treated identically, except for the use of the embedded gamified feedback acquisition mechanism that was removed from the control group.

5.3 Experiment Process

The planning process for the experiment was used as a checklist and guideline of what tasks had to be done. While conducting an experiment, planning is crucial to ensure that the results of the experiment become beneficial. Poor planning may ruin any well-intended study [13].

The experiment process was divided into six phases. The whole procedure was partly iterative and it would have been possible to go back and refine a previous activity before continuing with the final experiment. Especially

with the execution of the pilot experiment, which provided crucial evidence that were used to refine and improve the design of the final experiment. The following image illustrates the sequence of the different phases of our research (*Figure 5.2*).

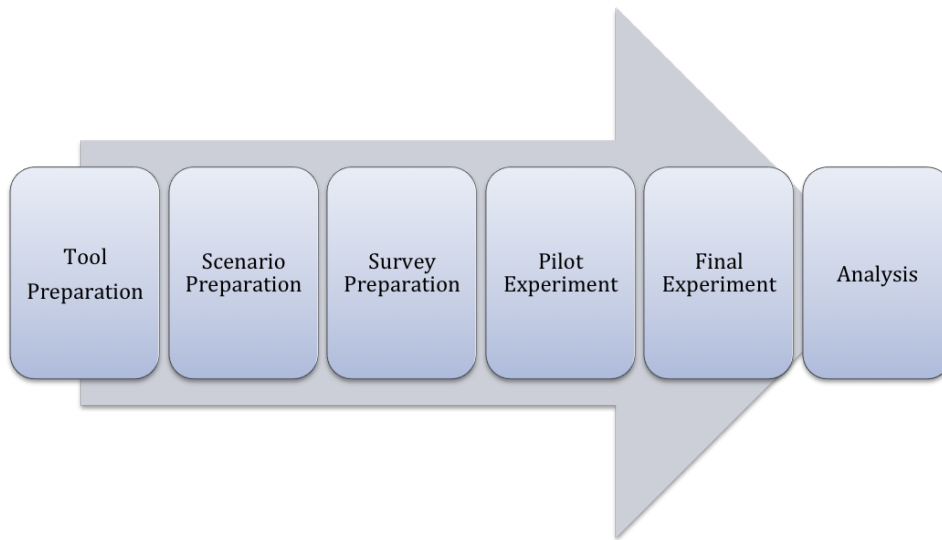


Figure 5.2: Experiment phases.

5.3.1 Tool Preparation

JEdit with Embedded Gamified Feedback Acquisition

To generalize the results, Wohlin in his work [13] suggests to execute the experiment within the environment of a real software application, which explains our choice of using an already existing application. We prepared an already existing the text editor, “JEdit” which is developed under the GNU General Public License³ and by using the Java programming language. This ensured that it was possible to get our hands into JEdit’s source code and that our tool will be able to be used along with JEdit. We imported the source code of JEdit into the Eclipse IDE and we identified the key classes that were responsible for the user interface of the application and it’s functionality.

Our first activity is to tweak the functionality of certain JEdit’s user interface elements. We affected the functionality of opening and saving a document, we reversed the functionality of undo, redo, cut, copy and paste buttons, bold, italic, underline buttons keyboard combinations were

³www.gnu.org/licenses/gpl-3.0.en.html

altered and the functionality of the “new line” (ENTER on the keyboard) was adjusted to exhibit faulty behavior 40% of the times it was pressed. These adjustments to the functionality of JEdit were intentional, in order to introduce some bugs in activities the participant was more probable to execute. This was a way to “provoke” some feedback from the users in order to ensure that eventually they will experience our embedded gamified feedback acquisition tool or the *Mantis Bug Tracker*.

At this point we took the role of a JEdit’s developer that would like to enable the embedded gamified feedback acquisition into JEdit. We firstly provided our preferences to the tool’s wizard by identifying the code areas to be affected by our tool, defining the feedback button position, the feedback types to be acquired and the gamification elements to be used for motivating the users.

After the application of our tool, all the user interface elements still have the functionality the used to have before but also they allow feedback to be acquired for each one of them. The user is able to report feedback for the application as a whole by pressing the feedback button, positioned in the upper right corner of the user interface screen; but also he or she can report feedback for each certain user interface element by hovering over the mouse cursor over that element and pressing a certain key combination (“CTRL+F”) (*Figure 5.3*).

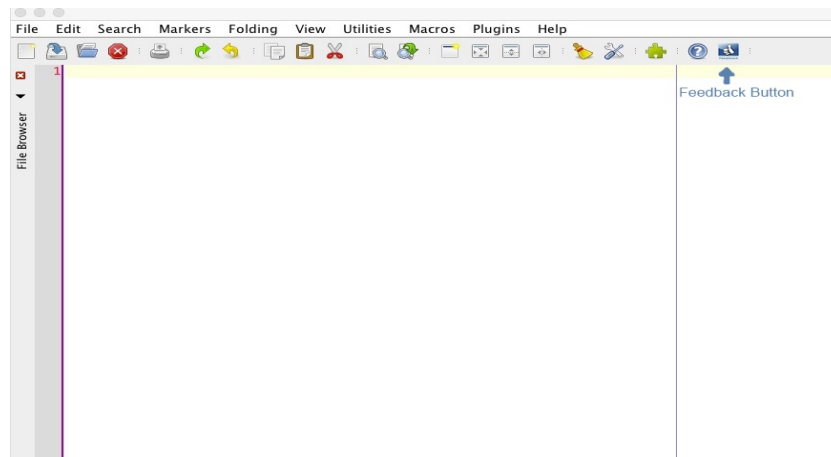


Figure 5.3: JEdit environment.

As soon as the user decides to send his or her feedback for the application or a certain user interface element, he/she is being redirected to an on-line feedback acquisition form, where four different types of feedback are

being acquired. These are “General” remarks, “Comments”, “Bug Reports”, and “Feature Requests”. These four feedback types were chosen as the most broadly used according to our explorative research and the scientific literature.

The user is also able to provide information about the severity and the reproducibility of the issue he or she wants to report and provide a description. All the reported feedback is being gathered in an SQL⁴ (Structured Query Language) Database which is designed and created according to the “Mantis Bug Tracker” tool standards (*Figure 5.4*).

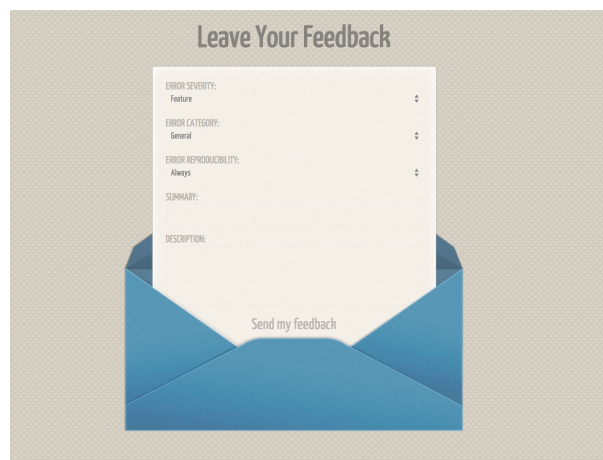


Figure 5.4: On-line feedback acquisition form.

As a final step, upon the completion of reporting a feedback remark, the user is presented with a point system and a reward list which is correlated to the points earned. Each time the user provides one feedback remark, ten points are being awarded as part of the effort to motivate the user. The idea is based on the extrinsic gamification principles according to the literature and was used as a simple example for demonstrating our tool. Of course more complex and maybe more effective intrinsic gamification elements could be used along with our tool in order to achieve better results towards user engagement (*Figure 5.5*).

The whole process is iterative, meaning that the end user is able to provide his or her feedback as many times as he or she wants in order to collect more points and try to obtain the top reward. This would serve as the first version of our experiment tool, which was going to be experienced by our treatment group participants.

⁴en.wikipedia.org/wiki/SQL

Thank you for your feedback!

Points	Rewards
200	20 Euro Itunes Card
150	Cup of Coffee & cookie
100	A cookie
50	Mars Bar
Your Points: 10	

Figure 5.5: Points - Rewards table.

JEdit with Embedded Mantis Bug Tracker

For the second version of our experiment tool we had to provide a different already existing feedback acquisition tool. *Mantis Bug Tracker* is the tool of our choice mainly because is a broadly used feedback acquisition tool by open source software developers.

The functionality of JEdit in this version of the experiment was exactly the same as the already aforementioned one. The main difference was that when the end users wanted to provide his or her remarks by pressing the feedback button (or the “CTRL+F” keyboard combination), he or she was being redirected to the on-line form of the *Mantis Bug Tracker*. The feedback provided by using *Mantis Bug Tracker* was stored in a similar design database as with our tool which consisted of different tables. That way we made sure there is no confusion between the remarks provided by both the treatment and the control group (*Figure 5.6*).

Moreover, in this version, after the completion of a reported remark by the user, the point rewarding system was absent in order to eliminate any presence of gamification elements.

The whole process was also iterative, meaning that the end user was able to provide his or her feedback as many times as he or she wanted. This would serve as the second version of our experiment tool, which was going to be experienced by our control group participants.

Enter Report Details	
Category	(select)
Reproducibility	have not tried
Severity	minor
Priority	normal
Select Profile	
Or Fill In	
Platform	
OS	
OS Version	
Summary	
Description	

Figure 5.6: Mantis Bug Tracker on-line form.

5.3.2 Scenario Preparation

In order to provide an activity for the participants we created the following scenario. A printed paper description was handed into every participant which included the steps to be taken. Each participant had to start the JEdit application, open a text file named in a specific way and start typing the text provided to them in printed paper. The text was long enough, thus they were asked to spend some time typing but they did not have to complete the whole document. The text provided is written in such a way, that eventually the participants would have to use certain user interface elements whose functionality was adjusted. This decision, is based on the fact that the user definitely will experience some flaws and bugs while typing.

On the printed paper description it is clearly stated that it is possible to provide some feedback concerning the JEdit text editor and help the development team of the application and the participants are free to do so if they chose to (*Figure 5.7*).

Additionally, to the experiment instructions and the text to be typed on JEdit, a printed paper sheet is also distributed to every respondent. This sheet includes the functionality of the user interface elements that might be useful while typing the requested text, along with the functionality of the feedback button (*Figure 5.8*).

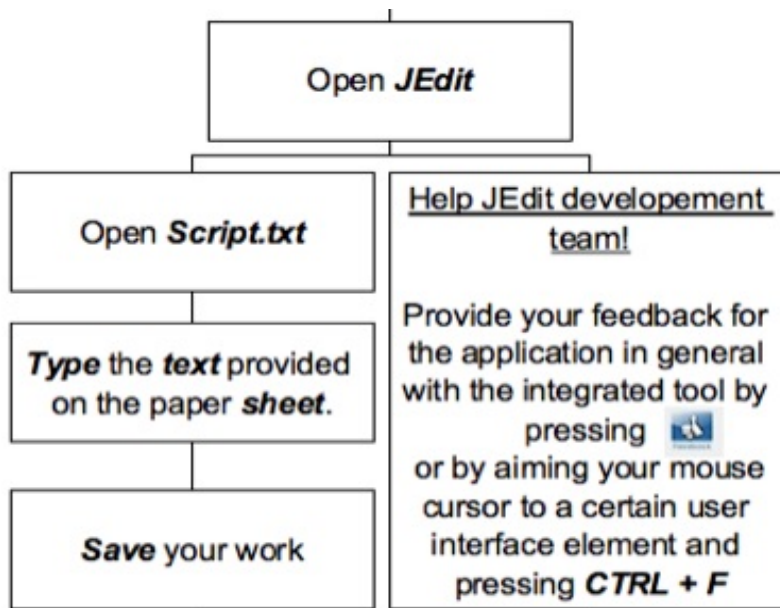







Figure 5.7: Experiment instruction sheet.

5.3.3 Survey Preparation

An on-line survey tool “*Web Survey Creator*”⁵, was used to create and present the experiment SUS questionnaire. Firstly, an introduction page was presented as an explanation of the following experiment and its duration. Before starting with the questionnaire, subjects were presented with an explanation of software feedback acquisition and the reasons we are exploring different aspects of feedback acquisition. Furthermore, the participants were notified about the software application (JEdit) they were going to use and a paper printed version of the scenario and JEdit’s functionality was distributed to them. The page to follow included a demographic questionnaire in order to identify certain personal information about the participants who were informed that their personal details would be treated confidentially. Moreover, a graphical illustration of the survey sequence was presented afterwards in order to clarify the next steps of the experiment. Last but not least, the ten questions of the SUS test were included in order for the participant to provide his or her evaluation of the tool he just used. The questions provided five response options for the respondents; ranging from “*Strongly Disagree*” to “*Strongly Agree*”. Additionally, the last page of the survey was offering space for any comments or additional remarks any participant would like to share.

⁵<http://www.websurveycreator.com/>

<u>Functionality</u>	<u>Key Combination/ Symbol</u>	
BOLD	CMD + B (Bold Dropdown Menu)	
ITALIC	CMD + B (Italic Dropdown Menu)	
UNDERLINE	CMD + B (Underline Dropdown Menu)	
CUT		
COPY		
PASTE		
SAVE		
FEEDBACK*	CTRL + F	
NEW LINE	ENTER OR SHIFT + ENTER	

*Aim your mouse pointer on any element on JEdit user interface and press CTRL + F to provide your feedback for that certain element.

Figure 5.8: JEdit functionality sheet.

5.3.4 Pilot Experiment

After preparing the application to be used and the survey, it was beneficial to execute a preliminary experiment. This ensured the evaluation feasibility, necessary time, design problems, select an appropriate sample size and improvement upon the experimental design. Thus, an internal employee of the University of Utrecht and two fellow colleagues were asked to spend some time on our existing experimental setup and go through the whole experiment process.

As soon as the process was complete, the participants were interviewed and were asked to denote their feedback for possible improvements. After running the preliminary experiment and going through the participants answers, we identified whether we could extract the information we need by this type of setup or some alterations had to be introduced. We identified by their feedback that we should perform the following alterations:

- A more concrete, clearer printed description of the experiment procedure should be added.
- Some bugs introduced in JEdit were quite “annoying”.
- Different script to be used (preferably a famous song’s lyrics).
- Functionality of JEdit should be better documented.

According to the pilot experiments participant’s feedback, we redefined several aspects of our experiment. We included paper printed experiment description along with the functionality capabilities of JEdit. This paper version was handed in the final experiment participants. We did use a famous song’s lyrics as the text to typed by the participants during their interaction with JEdit. And we did reduce the faulty behavior of the “ENTER” key to 60% and added a new keyboard combination “SHIFT+ENTER” for the new line character which was always functioning.

For the preliminary experiment phase both our embedded gamified feedback acquisition tool and *Mantis Bug Tracker* were tested in order to have a clear overview of how participants would interact with both versions of the experiment.

5.3.5 Final Experiment

As soon as the appropriate changes were applied to our experimental setup, the final experiment was executed. The experiment sessions took place in the environment of Utrecht University Library, where people were kindly asked to participate voluntarily in our study. Participants were international students between the ages of twenty and thirty years old.

Firstly, the treatment group sessions were performed by involving the first ten participants. As soon as the the data concerning the the embedded gamified feedback version were gathered, we continued by performing the control group sessions. Those sessions took place also in the same library environment and involved ten more different respondents, who experienced the same experiment process but instead of using the embedded gamified feedback version; they utilized the *Mantis Bur Tracker* tool.

Both groups were notified beforehand that the whole experiment process would take around twenty five minutes to complete, but they were allowed to spend more time in case they wanted to. During the experiment, there was no or very low interference with the subjects, just for explanation and clarification purposes towards the tasks of the experiment. This choice was part of our experimental design, because we did not want to affect in any way the respondents intention towards providing their feedback while using both of the tools provided.

5.3.6 Analysis

Last but not least, an appropriate statistical evaluation was used in order to compare the produced data from the two groups. We used a significance level of $p = 0.05$, two-tailed, due to the fact that we did not involve too

many participants in our experiment but we wanted to test if our tool is more effective than current existing feedback acquisition practices (*Mantis Bug Tracker*). If the observed sample result was below the significance level, then we could conclude that the observed effect actually reflects the characteristics of the population rather than just sampling error [4].

5.4 Threats to validity

While considering the planning phase of our experiment, a major concern was also how valid the outcome and the results from this experiment could be [13]. In order to address the validity of this study, the factors that may affect it had to be thoroughly determined. According to Wohlin's work [13], the methods used and the reasoning based upon the discovered information should be examined. Moreover, we also had to examine whether the conclusions drawn while interpreting the results of the study, could be generalized.

In this study we are examining the influence on the engagement and motivation of end users in the feedback acquisition process. Thus, feedback reports from our embedded gamified feedback acquisition mechanism are measured against the *Mantis Bug Tracker* tool. There is always the chance that the difference in the feedback reports between the two tools might have been affected by another unknown factor. Although, using a control group already ensures that our results stand up to meticulous examination [13].

In this research the following threats to validity were identified. These were the selection of gamification elements, instrumentation and selection of subjects. The construction of the experiment and its operation considered these threats and tried to avoid them. Since this research object is of artificial nature, it might be inaccurate or flawed. Aberrations and misleading conclusions might be drawn by using the artifact ineffectively. To avoid some errors a preliminary study was conducted to improve the experimental design.

The poor question expression and not the appropriate instrumentation could have had a negative effect on the reliability of the measures. To avoid this risk we decided to use the SUS questionnaire for examining the usability of both tools, which is broadly used and ensures high reliability and validity results.

The gamification elements were retrieved by identifying elements used in already existing research while exploring the field of gamification. The selection and implementation of the elements that were used in the gamified

feedback acquisition mechanism might not have the most appropriate ones to trigger motivation of the users.

Moreover, we would like to state that the results from our experiment are valid outside the real context in which the experiment was executed. However, there were some potential problems which could threaten the validity of the results. During an experiment people find themselves in an unfamiliar situation, making it difficult to guarantee that the outcome is caused by the intervention [2]. This is most probable due to the reason that people were interacting with a personal computer that was provided for them but it was not their own personal device.

Furthermore, there were some limitations to our experimental condition. First of all, our sample size was relatively small to make significant conclusions [4]. The reason was because we did rely on people's desire and voluntariness to participate in our study.

A further threat to validity was the possible interference between the experimenter and the participants. We did not want to affect the participant's desire to keep experimenting with JEdit and provide their remarks in any way. To avoid this problem we decided to have as less as possible interference and communication with the subjects and we only provided guidelines and directed the sequence of the experiment in case there was something not very clear to them.

Chapter 6

Results

Before the operation of the experiment was performed, all participants were briefed and handed the exact same instruction guidelines. The execution of the experiment went quite smooth, without any issues for both groups. In general barely any interaction was required by the researcher with only a few exceptions, where some clarifications were requested.

After the experiment ended each participant of the treatment group was receiving the reward he or she managed to achieve while providing his or her feedback. For the participants who managed to achieve the top score and obtained the top reward, were contacted via e-mail and the the reward was distributed to them electronically. Only the participants of the treatment group were received any rewards since they were the ones who experienced the embedded gamified feedback version of the experiment. Despite the same objective that was given to both groups, there was a quite distinctive difference between the remarks reported by the treatment and the control group.

Under the scope of our experiment, we decided to perform a statistical analysis by comparing the treatment and the control groups with students t-test. The type of t-test was unpaired and two tailed. Since the participants in both groups are different, unpaired t-test was used for statistical comparison. Before the collection of the data, since we did not predict which of the two groups would produce a larger mean, we performed a two tailed t-test with 95% ($p = 0.05$) confidence interval of the difference.

During the process of analyzing the data with the use of t-test we identified that, a parametric test such as the t-test is not appropriate for our analysis, therefore further investigation was required. A non-parametric test such as Mann-Whitney test was used, since the data points in the two groups being compared did not have equal variance. Equal variance between two groups

is compared with Levene median test and if the P-value is <0.05 , it implies that the data in the two groups being compared do not have equal variance.

The following parts of this chapter represent the data from the two versions, which were statistically analyzed and visualized with appropriate illustrations. The data to assess the quality and quantity of the feedback remarks reported were further evaluated by a separate statistical analysis. For simplicity reasons from now on we are going to refer to the embedded gamified feedback acquisition tool as the gamified tool and the *Mantis Bug Tracker* tool as the non gamified tool.

6.1 SUS Results

The first thing we examined from the data we gathered, were the respondent's answers on the SUS questionnaire which was part of the survey. We analyzed the responds on the ten questions of the SUS questionnaire for both the treatment and the control group.

To accomplish that, all participants answers were mapped onto a likert-scale ranging from 1 to 5 (with 1 representing "Strongly Disagree" and 5 representing "Strongly Agree"). You can find an example of the questions included in the SUS questionnaire in (*Figure A.1*) on the "Appendix A" but also our SUS questionnaire in (*Figure 6.1*). Additionally the participant's answers from both groups to the SUS questions can be found also on the "Appendix A" (*Table A.1*), (*Table A.2*).

6.1.1 Embedded Gamified Feedback Tool

By mapping the respondents answers onto likert-scale values, we identified the percentages of each choice per SUS question. In (*Figure 6.2*) it is possible to notice the responses provided by the treatment group, which experimented with the embedded gamified feedback tool.

By examining the the percentage rates on the participant's responses and by taking into account each SUS question, we can state that participants were confident using our embedded gamified feedback tool, did not too much or any guidance before starting using the tool. As one stated on his or her comments towards our tool, "*Well structured, not complex with unnecessary features - - to the point.*".

A fact that could be supported by the calculation of the SUS scores for every participant. SUS yields a single number representing a composite measure of the overall usability of the system being studied. Note that

I think that I would like to use this JEdit feedback tool frequently.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I found this JEdit feedback tool unnecessarily complex.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I thought this JEdit feedback tool was easy to use.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I think that I would need assistance to be able to use this JEdit feedback tool.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I found the various functions in this JEdit feedback tool were well integrated.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I thought there was too much inconsistency in this JEdit feedback tool.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I would imagine that most people would learn to use JEdit feedback tool very quickly.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I found this JEdit feedback tool very cumbersome/awkward to use.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I felt very confident using this JEdit feedback tool.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

I needed to learn a lot of things before I could get going with this JEdit feedback tool.

Disagree Strongly Disagree Indifferent Agree Agree Strongly

Figure 6.1: SUS questionnaire.

scores for individual items are not meaningful on their own. To calculate the SUS score, we first summed the score contributions from each item. Each item's score contribution will range from 1 to 5. For items 1,3,5,7 and 9 the score contribution is the scale position minus 1. For items 2,4,6,8 and 10, the contribution is 5 minus the scale position. Finally we multiplied the sum of the scores by 2.5 to obtain the overall value of SUS. SUS scores have

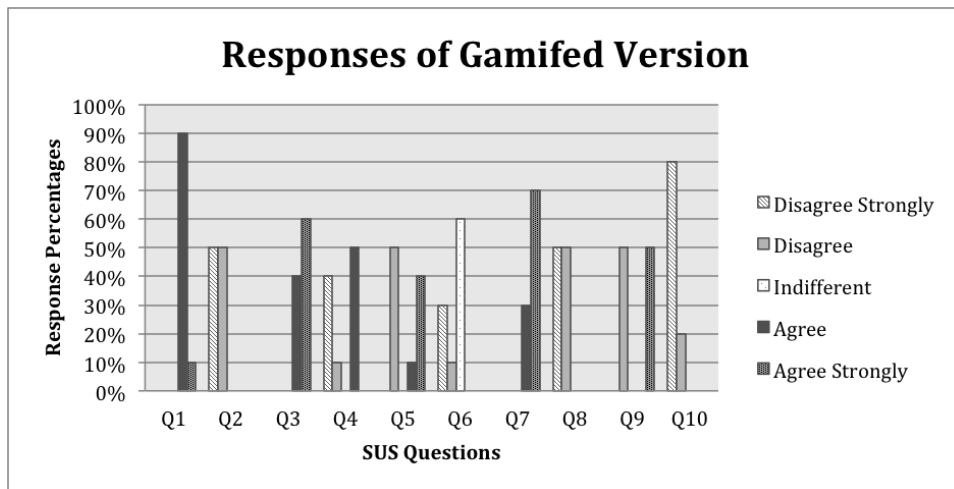


Figure 6.2: SUS questionnaire responses of the treatment group.

a range of 0 to 100 (the higher, the better).

On *Table 6.1*, we can observe the SUS scores for every participant who experienced the embedded gamified feedback tool. On the bottom line of the table, we average all the participant’s scores, which provides us with a score of 78 for the usability of our tool.

Once we have set a threshold significance level ($p = 0.05$), every result leads to a conclusion of either ”statistically significant” or not ”statistically significant”. Some statisticians feel very strongly that the only acceptable conclusion is significant or ’not significant’, and oppose use of adjectives or asterisks to describe values levels of statistical significance. Many scientists are not so rigid, and so prefer to use adjectives such as very significant or extremely significant. We are going to use this approach as shown in the *Figure 6.3* for denoting the statistical significant difference on the comparison of the answers between the two groups.

P value	Wording	Summary
< 0.001	Extremely significant	***
0.001 to 0.01	Very significant	**
0.01 to 0.05	Significant	*
>0.05	Not significant	ns

Figure 6.3: Statistical Significant Difference notations.

Participant	SUS Scores
1	67.5
2	92.5
3	62.5
4	92.5
5	62.5
6	95
7	67.5
8	90
9	60
10	90
Average	78

Table 6.1: SUS scores retrieved from the treatment group.

6.1.2 Embedded Mantis Bug Tracker Tool

We followed the same procedure for analyzing the results of the answers of the control group. We identified the percentages of each choice per SUS question. On (Figure 6.4) it is possible to notice the responses provided by the control group, which experimented with the *Mantis Bug Tracker* tool.

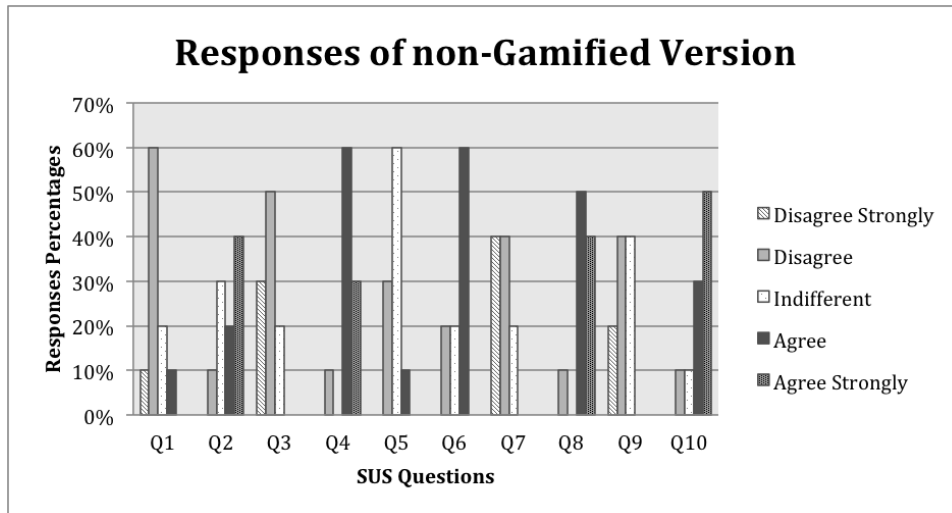


Figure 6.4: SUS questionnaire responses of the control group.

In Table 6.2, we can observe the SUS scores for every participant who experienced the *Mantis Bug Tracker* tool. On the bottom line of the table, we average all the participant's scores, which provides us with a score of 30.2

Participant	SUS Scores
1	30
2	15
3	37.5
4	37.5
5	42.5
6	27.5
7	40
8	12.5
9	30
10	17.5
Average	30.2777

Table 6.2: SUS scores of the control group.

for the usability of this tool, a score which is quite lower than the average outcome of our tool.

In *Figure 6.5* we can observe the average response scores as those were provided by the respondent's answers. Depending on the nature of the questions consisting the SUS questionnaire we can understand that, in the questions 1,3,5,7,9 the highest score provides a positive response. On the other hand, on the questions 2,4,6,8,10, the lowest score provides a positive response.

By identifying the average scores on the participant's responses we can state that the embedded gamified feedback tool (*gamified version*), acquired the most positive responses for most of the questions asked. An exception to this observation is the questions N.5, on which the responses average values are quite similar for both versions.

Moreover, in *Figure 6.5* we can appreciate that the gamified version's responses are significantly positive compared to the non-gamified version's responses. Statistical significance is denoted by asterisks along with the exact p values. Firstly, the responses' equality of variance was examined by executing the *Levene Test*.

The *Levene Test* provided us with a significance value to test the assumption of the equality of variance. If this significance value is greater than $p = 0.05$, then we did continue by executing an unpaired students t-test, two tailed in order to identify the statistical significance between the responses. Otherwise, if the *Levene significance value* was lower than $p =$

0.05, we continued by executing the Mann-Whitney test in order to identify the statistical significance between the responses.

We discovered that most of the SUS questionnaire answers are statistically significantly different according to both groups participant's remarks (*Figure 6.5*). Participants of the treatment group agreed significantly more to the fact that they would like to use the embedded gamified feedback acquisition mechanism compared to the *Mantis Bug Tracker* users (SUS Question 1, $p = 0.00012$).

The embedded gamified feedback acquisition mechanism was found to be significantly less complex by the treatments group's respondents compared to the *Mantis Bug Tracker*, which was used by the control group (SUS Question 2, $p = 0.0004$). Moreover, the embedded gamified feedback acquisition tool was denoted as significantly easier to use by respondents, compared to the *Mantis Bug Tracker* tool (SUS Question 3, $p = 0.000002$).

Treatment groups participants stated that the support of a technical person was not needed, in order to be able to use the tool, which was not the case for the *Mantis Bug Tracker* tool (SUS Question 4, $p = 0.0023$). Concerning the inconsistency of the tool, treatments group's participants found the embedded gamified feedback acquisition tool to be significantly less inconsistent compared to the *Mantis Bug Tracker* (SUS Question 6, $p = 0.0013$).

While the learn ability of the embedded gamified feedback acquisition tool was considered to be significantly higher than the *Mantis Bug Tracker* according to its users (SUS Question 7, $p = 0.0000001$). Last but not least, the embedded gamified feedback acquisition tool was considered to be significantly less cumbersome to use, while the treatment group's participants stated that they did not need to learn a lot of things before they could use the tool (SUS Question 8 and 10, $p = 0.00004$ and $p = 0.0000001$ respectively).

For the SUS questions 5 and 9 no statistical significance was detected after executing the Mann-Whitney test. Thus, we can state that both tools were equally well integrated to JEdit's environment and that treatment group's participants felt more (but not significantly) confident using the tool compared to the *Mantis Bug Tracker*. Statistical tests executed for each questions are enclosed on "Appendix A", (*Figures A.6-A.21*)

6.2 Feedback Remarks Acquired

On the following figures we provide the comparison of the means of the quantities of the general remarks, comments, bug reports and feature requests as those were expressed by participants of both groups. The error bars illustrate the standard deviation of the answers on each feedback type acquired, and the asterisks illustrate the statistical significant difference according to the notation of *Figure 6.3*.

Overall average quantity of all feedback types measured is greater while using the embedded gamified feedback acquisition tool compared to the *Mantis Bug Tracker*. The statistical analysis followed was the same as previously executed for the SUS data and provides us with the statistical significant differences ranging from $p = 0.0000005$ for feature requests, $p = 0.000022$ for bug reports, $p = 0.000025$ for comments and $p = 0.0015$ general remarks provided by the experiment participants.

At this point it should be denoted that we measured the quality of the different feedback types reported by both group's participants, by creating a scale from 1 to 3 to define each remark's quality. "Poor" quality is mapped to 1, "fair" quality to number 2 and "good" quality to number 3. This scale and the definition of each reported remarks quality is decided by us while identifying each participant's feedback report.

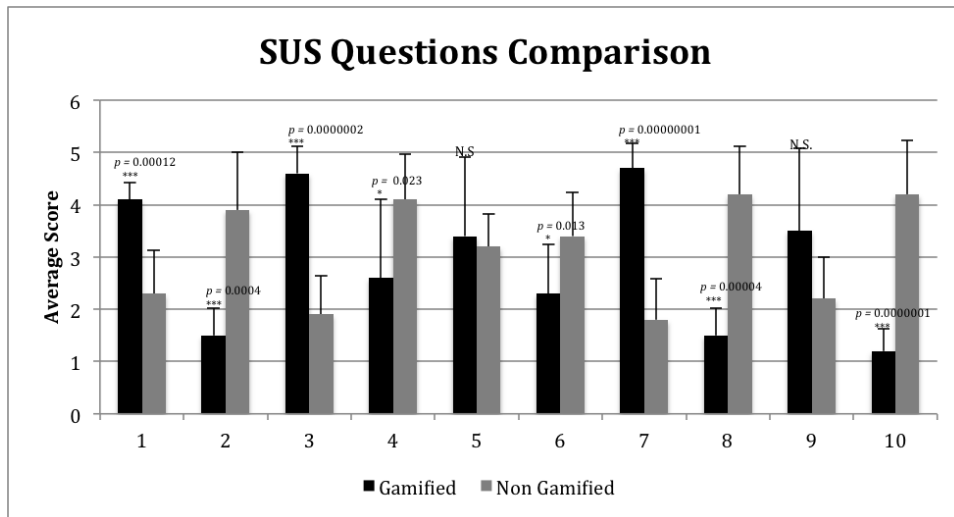


Figure 6.5: Average response scores provided by treatment and control groups and statistical significant difference.

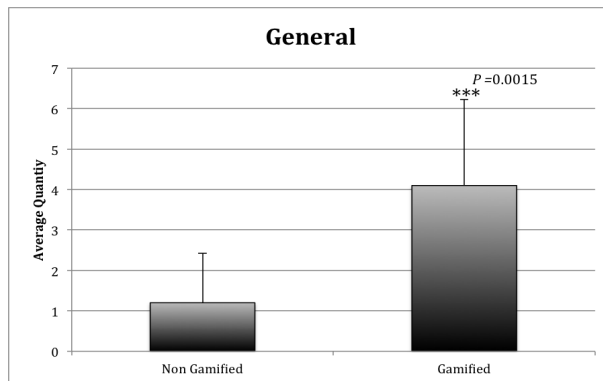


Figure 6.6: Average quantity general remarks provided by treatment and control groups. $P = 0.0015$

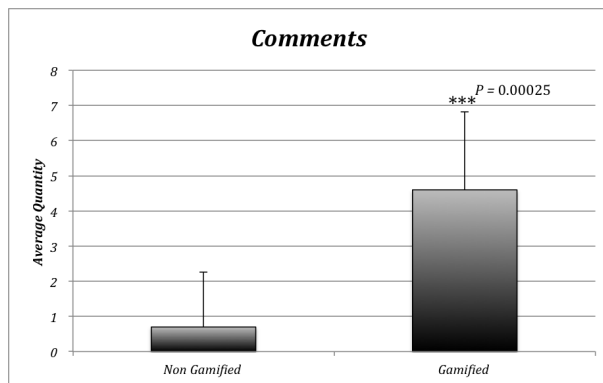


Figure 6.7: Average quantity comments provided by treatment and control groups. $P = 0.00025$

The overall average quality of all feedback types reported is quite similar amongst the two tools. The statistical analysis followed was the same as previously executed for the feedback type quantity and provides us with the statistical significant differences ranging from $p = 0.00013$ for comments and $p = 0.0043$ general remarks. There was no statistical significant difference identified for the quality of the bugs and feature requests reported (*Figure 6.10*).

6.3 Discussion

From the examination of the results provided by the experimental study it is evident that the two experimental groups exhibited different behaviors while interacting with both the gamified and the non gamified tools. Both the treatment and the control groups experimented on the same library

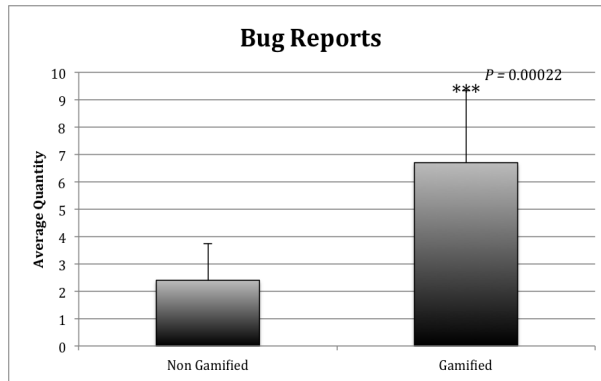


Figure 6.8: Average quantity bugs provided by treatment and control groups. $P = 0.00022$

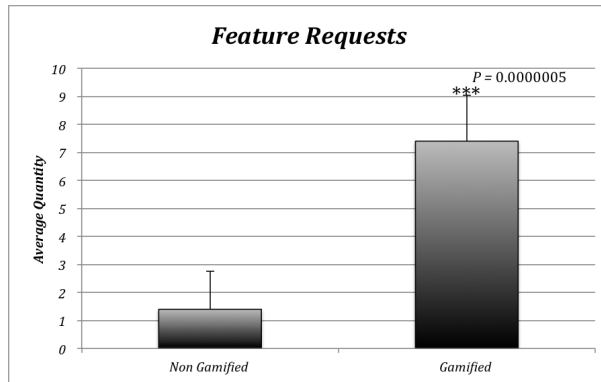


Figure 6.9: Average quantity feature requests provided by treatment and control groups. $P = 0.000005$

setting environment, received the same instructions towards the execution of the experiment and used the exact same hardware and software.

The main difference was the feedback acquisition mechanism which was embedded into JEdit's environment. This mechanism served as the mean for the participants to report any kind of feedback remarks while they were interacting with JEdit application. The treatment group experienced a simple acquisition platform enhanced with a certain gamification element, while the control group was able to report any feedback remarks by using an existing feedback acquisition tool, the *Mantis Bug Tracker*.

6.3.1 Framework Usability

By observing the graphical illustrations, we can state that the treatment group was more satisfied by the feedback acquisition mechanism provided for

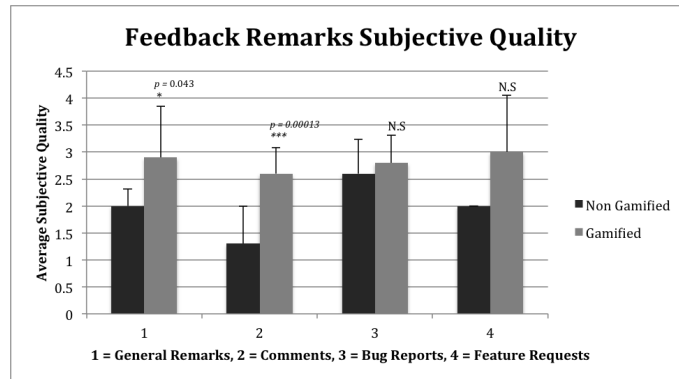


Figure 6.10: Average subjective quality per feedback type reported. Significance levels denoted with asterisks.

them compared to the control group (*Section 6.1.1*). Such statement can be expressed by firstly examining the SUS scores of both feedback acquisition mechanisms and observe that the treatment group (*Table 6.1*) provided higher score (*average score value 78*) on the mechanism they used, compared to the one experienced by the control group (*average score value 30.27*), (*Table 6.2*).

Moreover, not only the SUS high score of the treatment group allows us to understand that this groups participant's were more satisfied, but also that the tool was rated as more usable than the *Mantis Bug Tracker* tool. A fact, supported by most of the participant's answers on the SUS questions concerning the complexity, the usability and the background knowledge required for the use of that certain feedback acquisition mechanism.

On the other hand, the control group's SUS scores provided to the *Mantis Bug Tracker*, allow us to assume that the participants were not satisfied enough with the tool in order to evaluate it with higher usability ratings. Most of the SUS questions which concerned the usability and the ease of use towards the feedback reporting process, were lower rated compared to our developed tool. A case that can be also be supported by the fact that, during the experiment process, there were a few individual cases where the respondents chose to report no feedback when using the tool.

Nonetheless, both feedback acquisition mechanisms were identified as being adequately integrated into JEdit's environment, a conclusion that is drawn from the equal positive rating on the fifth question of the SUS questionnaire. This observation also increases our confidence about integrating both feedback acquisition mechanisms in the same manner into JEdit's environment. This allows us to safely assume that the main contribution to

the difference of the usability ratings lies in the different “nature” of the tools and not its way of integration.

Based on the statistical analysis performed for the SUS questionnaire data and *Figure 6.4* we can state that, participants interacting with the embedded gamified feedback acquisition version; significantly provided higher scores concerning the usability of the tool. Thus, the embedded gamified feedback acquisition version appears to be significantly more usable than the *Mantis Bug Tracker* tool for the feedback acquisition process.

6.3.2 Feedback Remarks Acquired

Moreover, while executing our result’s analysis we identified that there was significant difference on the quantity and the quality of the feedback remarks reported by participants of both groups. First of all, the average number of general remarks, comments, bug reports and feature requests provided by the treatment group was significantly higher than those provided by the control group. As presented in the graphical illustration in Chapter 6 (*Figure 6.6-6.9*). The evidence presented in *Figures 6.6-6.9* can be supported by displaying the data acquired in *Table 6.3*, which presents the average number of every feedback type reported by both groups, along with the standard deviation calculated and their statistical significant difference.

	General Remarks		Comments		Bug Reports		Feature Requests	
	Treatment Group	Control Group	Treatment Group	Control Group	Treatment Group	Control Group	Treatment Group	Control Group
Mean	4.1	1.2	4.6	0.7	6.7	2.4	7.4	1.4
Standard Deviation	2.13	1.22	2.22	1.56	2.62	1.34	1.64	1.34
Statistical Significant Difference	$p = 0.0015$		$p = 0.00025$		$p = 0.00022$		$p = 0.00000005$	

Table 6.3: Average quantity per feedback type reported by both groups, statistical significant differences.

Additionally, to the participant’s reported feedback quantity, we measured the quality of the remarks. In order to do that we analyzed the content of each feedback remark provided, and allocated a quality definition of poor, fair or good. The analyzed data illustrated in *Figure 6.10* in Chapter 6, provide the subjective quality for all four types of feedback acquired. The information presented in *Figure 6.10* can be supported by their representation in *Table 6.4*, which provides the average subjective quality of every feedback type reported by both groups, along with the standard deviation calculated and their statistical significant difference.

	General Remarks		Comments		Bug Reports		Feature Requests	
	Treatment Group	Control Group	Treatment Group	Control Group	Treatment Group	Control Group	Treatment Group	Control Group
Mean	2.9	2	2.6	1.3	2.8	2.6	3	2
Standard Deviation	0.31	0.94	0.69	0.48	0.63	0.51	0	1.05
Statistical Significant Difference	<i>p</i> = 0.0043		<i>p</i> = 0.00013		N.S.		N.S.	

Table 6.4: Average subjective quality per feedback type reported by both groups, statistical significant differences.

After acquiring the background knowledge concerning the experimental setup and after analyzing its results we can state that there was significant difference on the way participant's encountered the feedback acquisition process on both versions of the experiment. Experimental results along with the statistical analysis, support the fact that the gamified version enjoyed better usability ratings than the non gamified version; and led into higher amounts of feedback remarks reported with equal or better quality. We can safely express that the simplistic and user friendly environment of the gamified version along with the use of a certain extrinsic gamification element, provided quite a distinctive difference both for the usability of the tool and the quantity of the feedback reported.

Chapter 7

Conclusions, Limitations, Future Perspectives

7.1 Answers to the Research Questions

In the scope of this research, we provided some insights towards the design and implementation of an embedded gamified feedback acquisition mechanism. The experimental process showed that such a mechanism has the potential to be successful into the process of feedback acquisition. This success however, highly depends on the design choices of game mechanics and game elements, as they can affect different psychological needs. After analyzing the experimental results, it was identified that an embedded, simple, user friendly, gamified feedback acquisition mechanism can motivate and engage more actively the users towards the feedback gathering process.

The aim of this research was to find answers to our main research question, which was stated as follows:

- *How to enable the development of software systems where feedback acquisition and processing are integral components of the system to-be?*

In order to answer this main question, three sub-questions were formulated each addressing a different problem area. Based on the literature study and results derived from the conducted experiment, we were able to answer them with following statements.

Sub Questions:

- *Which are the appropriate mechanisms for effective feedback acquisition?*

After examining the related literature and by getting our hands on the current feedback acquisition practices, we identified that current feedback acquisition practices offer the foundations for feedback reporting. Nevertheless, a big gap still exists between the developers and the end users for whom software applications are designed and developed. In order to effectively acquire feedback, the approach of a feedback acquisition framework which will be embedded in already existing applications and will engage and motivate the users by exploiting the advantages of gamification elements should be adopted.

- *How to integrate such feedback mechanisms in a reusable manner into existing software development practices?*

In order to design and develop such a feedback acquisition framework, which will affectively acquire feedback and engage the end users, current software engineering practices should be adopted. Depending on the software applications the framework is addressing, appropriate framework requirements should be identified, a suitable programming language and IDE should be adopted for use and of course the appropriate game mechanics and gamification elements should be determined.

- *How can we engage and involve more users in a systematic way?*

The design of our experiment demonstrated that a gamified experience in the feedback acquisition process, is able to effectively influence users behavior. Not only did the treatment group provide more feedback remarks, but also with equal or higher quality compared to the control group. Moreover, such an enhanced feedback mechanism enjoyed higher usability ratings compared to an existing feedback gathering instrument.

7.2 Limitations

There are several limitations to this research project that relate to the rather explorative nature of the study. Gamification is still a relatively young field; thus, literature on the game elements and the effect of the game elements on feedback acquisition is limitedly yet available. Therefore, the game mechanics and gamification elements to be adopted constitute an area further to be explored. Moreover, this current study lacks generalizations of its findings due to the small sample size which was used in the experimental study. Additionally, we can not generalize our findings for other existing feedback acquisition tools since we only compared our tool to *Mantis Bug Tracker*. This is a research study currently exploring an uncharted area hence, there is definitely room for continuous and repeated experiments and research examinations. The results seem very promising, although it

is not clear if the gamification factor played the most significant role in the significant difference between the two tools. The simpler and the more user friendly interface of our tool may have affected in a positive way the participant's responses.

7.3 Future Work

This enigma could be answered by any related future investigations towards that direction. As in every scientific field, repetition in the experimental process is needed in order to identify a significant artifact. Thus, further investigation and experimentation with gamification principles should definitely be performed. Moreover, apart from the comparison with the *Mantis Bug Tracker* tool, our embedded gamified feedback acquisition mechanism can also be correlated against other existing feedback acquisition tools. This correlation could provide some more concrete findings about our tool's potential capabilities. Additionally to a future experimentation setup, an existing feedback acquisition mechanism could be used and that mechanism could be enhanced with game mechanics. This investigation could be probably bring to the surface some very interesting findings, concerning the significant role of gamification on the feedback acquisition process along with currently existing feedback acquisition techniques.

Appendix A

Appendix

		Strongly Disagree				Strongly Agree
1.	I think that I would like to use this website frequently.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.	I found this website unnecessarily complex.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3.	I thought this website was easy to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4.	I think that I would need assistance to be able to use this website.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5.	I found the various functions in this website were well integrated.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6.	I thought there was too much inconsistency in this website.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7.	I would imagine that most people would learn to use this website very quickly.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8.	I found this website very cumbersome/awkward to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9.	I felt very confident using this website.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10.	I needed to learn a lot of things before I could get going with this website.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure A.1: System Usability Scale Questionnaire Example.

I think that I would like to use this JEdit feedback tool frequently.	Agree	Agree	Agree	Agree Strongly	Agree	Agree	Agree	Agree	Agree	Agree
I found this JEdit feedback tool unnecessarily complex.	Disagree	Disagree Strongly	Disagree	Disagree Strongly	Disagree	Disagree	Disagree	Disagree Strongly	Disagree Strongly	Disagree Strongly
I thought this JEdit feedback tool was easy to use.	Agree	Agree Strongly	Agree	Agree Strongly	Agree Strongly	Agree Strongly	Agree	Agree Strongly	Agree	Agree Strongly
I think that I would need assistance to be able to use this JEdit feedback tool.	Disagree Strongly	Disagree Strongly	Agree	Agree	Agree	Disagree Strongly	Agree	Disagree Strongly	Agree	Disagree
I found the various functions in this JEdit feedback tool were well integrated.	Disagree	Agree Strongly	Disagree	Agree Strongly	Disagree	Agree Strongly	Agree Strongly	Disagree Strongly	Disagree	Agree
I thought there was too much inconsistency in this JEdit feedback tool.	Indifferent	Indifferent	Indifferent	Disagree Strongly	Indifferent	Disagree Strongly	Indifferent	Disagree Strongly	Indifferent	Disagree
I would imagine that most people would learn to use JEdit feedback tool very quickly.	Agree	Agree Strongly	Agree Strongly	Agree Strongly	Agree Strongly	Agree Strongly	Agree	Agree Strongly	Agree	Agree Strongly
I found this JEdit feedback tool very cumbersome/awkward to use.	Disagree	Disagree Strongly	Disagree	Disagree Strongly	Disagree	Disagree Strongly	Disagree	Disagree Strongly	Disagree	Disagree Strongly
I felt very confident using this JEdit feedback tool.	Disagree	Agree Strongly	Disagree	Agree Strongly	Disagree	Agree Strongly	Disagree	Agree Strongly	Disagree	Agree Strongly
I needed to learn a lot of things before I could get going with this JEdit feedback tool.	Disagree Strongly	Disagree Strongly	Disagree Strongly	Disagree Strongly	Disagree	Disagree Strongly	Disagree Strongly	Disagree Strongly	Disagree	Disagree Strongly

Figure A.2: System Usability Scale Questionnaire Answers - Treatment Group.

I think that I would like to use this Mantis feedback tool frequently.	Disagree	Disagree	Indifferent	Disagree	Agree	Disagree	Disagree	Indifferent	Disagree	Disagree Strongly
I found this Mantis feedback tool unnecessarily complex.	Indifferent	Agree Strongly	Agree	Indifferent	Indifferent	Agree Strongly	Disagree	Agree	Agree Strongly	Agree Strongly
I thought this Mantis feedback tool was easy to use.	Disagree	Indifferent	Disagree	Disagree	Disagree	Disagree Strongly	Indifferent	Disagree Strongly	Disagree	Disagree Strongly
I think that I would need assistance to be able to use this Mantis feedback tool.	Agree	Agree Strongly	Disagree	Agree	Agree	Agree	Agree	Agree Strongly	Agree	Agree Strongly
I found the various functions in this Mantis feedback tool were well integrated.	Indifferent	Indifferent	Indifferent	Agree	Indifferent	Indifferent	Disagree	Indifferent	Disagree	Disagree
I thought there was too much inconsistency in this Mantis feedback tool.	Agree	Agree	Indifferent	Agree	Disagree	Indifferent	Disagree	Agree	Agree	Agree
I would imagine that most people would learn to use Mantis feedback tool very quickly.	Indifferent	Disagree Strongly	Disagree	Indifferent	Disagree	Disagree Strongly	Disagree	Disagree Strongly	Disagree	Disagree Strongly
I found this Mantis feedback tool very cumbersome/awkward to use.	Agree	Agree Strongly	Agree	Agree	Disagree	Agree Strongly	Agree	Agree Strongly	Agree	Agree Strongly
I felt very confident using this Mantis feedback tool.	Disagree	Disagree Strongly	Indifferent	Indifferent	Disagree	Indifferent	Disagree	Disagree Strongly	Disagree	Indifferent
I needed to learn a lot of things before I could get going with this Mantis feedback tool.	Agree Strongly	Agree Strongly	Agree Strongly	Agree	Agree Strongly	Disagree	Indifferent	Agree Strongly	Agree	Agree

Figure A.3: System Usability Scale Questionnaire Answers - Control Group.

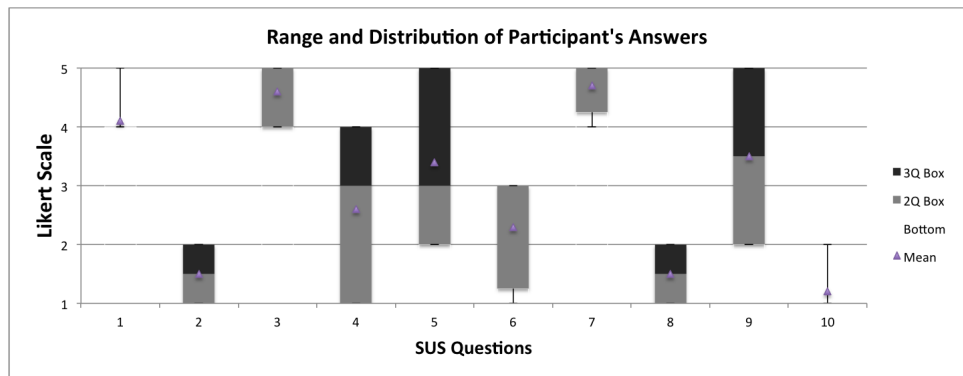


Figure A.4: Boxplot for the responses of the treatment group.

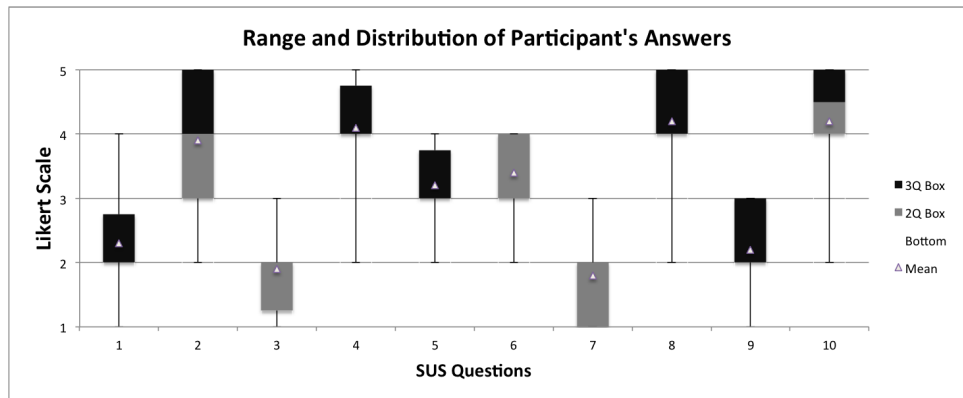


Figure A.5: Boxplot for the responses of the control group.

SUS Question 1 – T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	4.10	.316	.100
	Non Gamified	10	2.30	.823	.260

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means					95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	6.153	.023	6.454	18	.000005	1.800	.279	1.214	2.386
	Equal variances not assumed			6.454	11.599	.000037	1.800	.279	1.190	2.410

Figure A.6: SUS Question 1 T-Test.

Hypothesis Test Summary			
Null Hypothesis	Test	Sig.	Decision
1 The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	1.299E-4 ₁	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁ Exact significance is displayed for this test.

Figure A.7: SUS Question 1 Mann - Whitney Test.

SUS Question 2 – T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	1.50	.527	.167
	Non Gamified	10	3.90	1.101	.348

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means					95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	6.517	.020	-6.220	18	.000007	-2.400	.386	-3.211	-1.589
	Equal variances not assumed			-6.220	12.922	.000032	-2.400	.386	-3.234	-1.566

Figure A.8: SUS Question 2 T-Test.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	4.330E-5 ₁	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁ Exact significance is displayed for this test.

Figure A.9: SUS Question 2 Mann - Whitney Test.

SUS Question 3 - T-Test

Group	N	Mean	Std. Deviation	Std. Error Mean
FeedbackAcquisitionMethod Gamified	10	4.60	.516	.163
FeedbackAcquisitionMethod Non Gamified	10	1.90	.738	.233

		Levene's Test for Equality of Variances		t-test for Equality of Means					95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	.156	.698	9.480	18	.000000020	2.700	.285	2.102	3.298
	Equal variances not assumed			9.480	16.111	.000000054	2.700	.285	2.097	3.303

Figure A.10: SUS Question 3 T-Test.

SUS Question 4 - T-Test

Group	N	Mean	Std. Deviation	Std. Error Mean
FeedbackAcquisitionMethod Gamified	10	2.60	1.506	.476
FeedbackAcquisitionMethod Non Gamified	10	4.10	.876	.277

		Levene's Test for Equality of Variances		t-test for Equality of Means					95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	13.914	.002	-2.724	18	.014	-1.500	.551	-2.657	-.343
	Equal variances not assumed			-2.724	14.463	.016	-1.500	.551	-2.678	-.322

Figure A.11: SUS Question 4 T-Test.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	.023 ₁	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁ Exact significance is displayed for this test.

Figure A.12: SUS Question 4 Mann - Whitney Test.

SUS Question 5 - T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	3.40	1.506	.476
	Non Gamified	10	3.20	.632	.200

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	36.344	.000011	.387	18	.703	.200	.516	-.885	1.285
	Equal variances not assumed			.387	12.081	.705	.200	.516	-.924	1.324

Figure A.13: SUS Question 5 T-Test.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	.971 ₁	Retain the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁ Exact significance is displayed for this test.

Figure A.14: SUS Question 5 Mann - Whitney Test.

SUS Question 6 – T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	2.30	.949	.300
	Non Gamified	10	3.40	.843	.267

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
FeedbackAcquisitionMethod	Equal variances assumed	.573	.459	-2.741	18	.013	-1.100	.401	-1.943	-.257
	Equal variances not assumed			-2.741	17.756	.014	-1.100	.401	-1.944	-.256

Figure A.15: SUS Question 6 T-Test.

SUS Question 7 – T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	4.70	.483	.153
	Non Gamified	10	1.80	.789	.249

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
FeedbackAcquisitionMethod	Equal variances assumed	2.367	.141	9.915	18	.000000010	2.900	.292	2.285	3.515
	Equal variances not assumed			9.915	14.918	.000000059	2.900	.292	2.276	3.524

Figure A.16: SUS Question 7 T-Test.

SUS Question 8 – T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	1.50	.527	.167
	Non Gamified	10	4.20	.919	.291

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
FeedbackAcquisitionMethod	Equal variances assumed	.503	.487	-8.060	18	.00000022	-2.700	.335	-3.404	-1.996
	Equal variances not assumed			-8.060	14.343	.00000107	-2.700	.335	-3.417	-1.983

Figure A.17: SUS Question 8 T-Test.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	4.330E-5 ₁	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁Exact significance is displayed for this test.

Figure A.18: SUS Question 8 Mann - Whitney Test.

SUS Question 9 – T-Test

Group Statistics

Group	N	Mean	Std. Deviation	Std. Error Mean
FeedbackAcquisitionMethod Gamified	10	3.50	1.581	.500
FeedbackAcquisitionMethod Non Gamified	10	2.20	.789	.249

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	44.258	.00000305	2.327	18	.032	1.300	.559	.126	2.474
	Equal variances not assumed			2.327	13.219	.037	1.300	.559	.095	2.505

Figure A.19: SUS Question 9 T-Test.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	.143 ₁	Retain the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁Exact significance is displayed for this test.

Figure A.20: SUS Question 9 Mann - Whitney Test.

SUS Question 10 – T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	1.20	.422	.133
	Non Gamified	10	4.20	1.033	.327

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	5.492	.031	-8.504	18	.00000010	-3.000	.353	-3.741	-2.259
	Equal variances not assumed			-8.504	11.919	.00000210	-3.000	.353	-3.769	-2.231

Figure A.21: SUS Question 10 T-Test.

General Remarks Quantity – T-Test

[DataSet1] /Users/Illusioner/Documents/epesproject.sav

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	4.10	2.132	.674
	Non Gamified	10	1.20	1.229	.389

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	1.008	.329	3.727	18	.002	2.900	.778	1.265	4.535
	Equal variances not assumed			3.727	14.389	.002	2.900	.778	1.235	4.565

Figure A.22: General Remarks Reported Quantity T-Test.

General Remarks Quality – T-Test

Group Statistics										
Group		N	Mean	Std. Deviation	Std. Error Mean					
FeedbackAcquisitionMethod	Gamified	10	2.90	.316	.100					
	Non Gamified	10	2.00	.943	.298					

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	15.899	.001	2.862	18	.010	.900	.314	.239	1.561
	Equal variances not assumed			2.862	11.000	.015	.900	.314	.208	1.592

*Nonparametric Tests: Independent Samples.
 NPTESTS
 /INDEPENDENT TEST (FeedbackAcquisitionMethod) GROUP (Group)
 /MISSING SCOPE=ANALYSIS USERMISSING=EXCLUDE
 /CRITERIA ALPHA=0.05 CILEVEL=95.

Nonparametric Tests

Hypothesis Test Summary			
Null Hypothesis	Test	Sig.	Decision
1 The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	.043 ₁	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.
₁ Exact significance is displayed for this test.

Figure A.23: General Remarks Reported Quality T-Test and Mann-Whitney Test.

Comments Quantity – T-Test

Group Statistics					
Group		N	Mean	Std. Deviation	Std. Error Mean
FeedbackAcquisitionMethod	Gamified	10	4.60	2.221	.702
	Non Gamified	10	.70	1.567	.496

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	3.383	.082	4.537	18	.00025527	3.900	.860	2.094	5.706
	Equal variances not assumed			4.537	16.180	.00032786	3.900	.860	2.079	5.721

Figure A.24: Comments Reported Quantity T-Test.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	2.057E-4 ₁	Reject the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁ Exact significance is displayed for this test.

Figure A.25: Comments Reported Quantity Mann-Whitney Test.

Comment Quality - T-Test

Group Statistics					
	Group	N	Mean	Std. Deviation	Std. Error Mean
FeedbackAcquisitionMethod	Gamified	10	2.60	.699	.221
	Non Gamified	10	1.30	.483	.153

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	1.102	.308	4.837	18	.00013	1.300	.269	.735	1.865
	Equal variances not assumed			4.837	15.997	.00018	1.300	.269	.730	1.870

Figure A.26: Comments Reported Quality T-Test.

Bug Report Quantity - T-Test

Group Statistics					
	Group	N	Mean	Std. Deviation	Std. Error Mean
FeedbackAcquisitionMethod	Gamified	10	6.70	2.627	.831
	Non Gamified	10	2.40	1.350	.427

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	2.024	.172	4.604	18	.0002202	4.300	.934	2.338	6.262
	Equal variances not assumed			4.604	13.444	.0004534	4.300	.934	2.289	6.311

Figure A.27: Bugs Reported Quantity T-Test.

Bug Reports Quality - T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	2.80	.632	.200
	Non Gamified	10	2.60	.516	.163

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means					95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	.540	.472	.775	18	.449	.200	.258	-.342	.742
	Equal variances not assumed			.775	17.308	.449	.200	.258	-.344	.744

Figure A.28: Bugs Reported Quality T-Test.

Feature Requests Quantity - T-Test

Group Statistics					
Group	N	Mean	Std. Deviation	Std. Error Mean	
FeedbackAcquisitionMethod	Gamified	10	7.40	1.647	.521
	Non Gamified	10	1.40	1.350	.427

Independent Samples Test										
		Levene's Test for Equality of Variances		t-test for Equality of Means					95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	.167	.687	8.911	18	.00000005	6.000	.673	4.585	7.415
	Equal variances not assumed			8.911	17.334	.00000007	6.000	.673	4.582	7.418

Figure A.29: Feature Requests Reported Quantity T-Test.

Feature Requests Quality – T-Test

Group	N	Mean	Std. Deviation	Std. Error Mean
FeedbackAcquisitionMethod Gamified	10	3.00	.000	.000
FeedbackAcquisitionMethod Non Gamified	10	2.00	1.054	.333

		Levene's Test for Equality of Variances		t-test for Equality of Means						
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
									Lower	Upper
FeedbackAcquisitionMethod	Equal variances assumed	.000	.999	3.000	18	.008	1.000	.333	.300	1.700
	Equal variances not assumed			3.000	9.000	.015	1.000	.333	.246	1.754

*Nonparametric Tests: Independent Samples.
 NPTESTS
 /INDEPENDENT TEST (FeedbackAcquisitionMethod) GROUP (Group)
 /MISSING SCOPE=ANALYSIS USERMISSING=EXCLUDE
 /CRITERIA ALPHA=0.05 CILEVEL=95.

Nonparametric Tests

	Null Hypothesis	Test	Sig.	Decision
1	The distribution of FeedbackAcquisitionMethod is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	.063 ₁	Retain the null hypothesis.

Asymptotic significances are displayed. The significance level is .05.

₁Exact significance is displayed for this test.

Figure A.30: Feature Requests Reported Quality T-Test and Mann-Whitney Test.

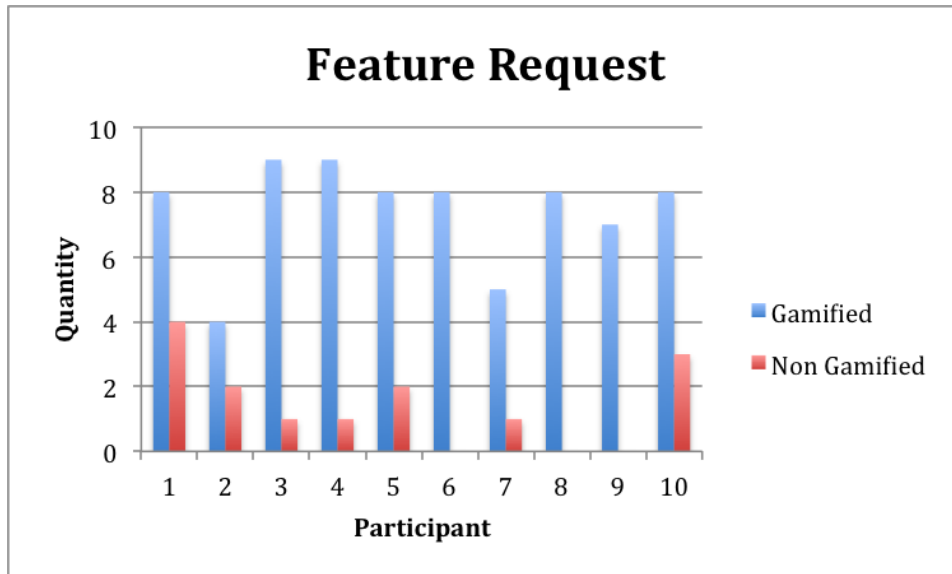


Figure A.31: Feature Requests reported by treatment and control group.

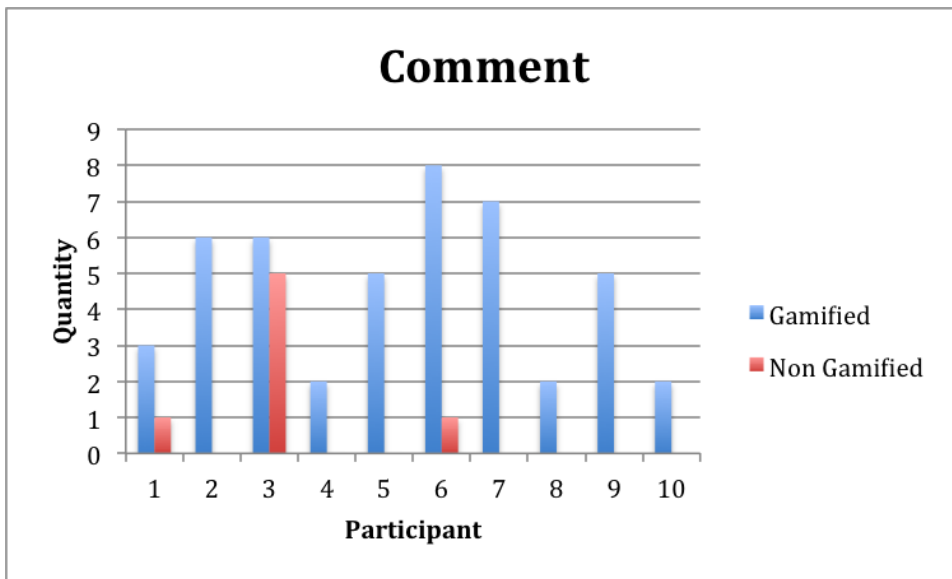


Figure A.32: Comments reported by treatment and control group.

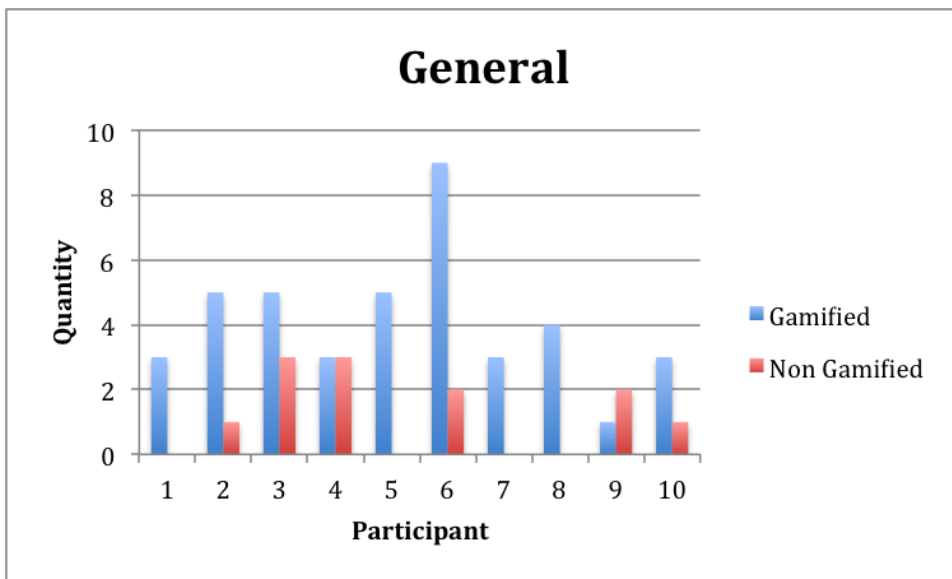


Figure A.33: General remarks reported by treatment and control group.

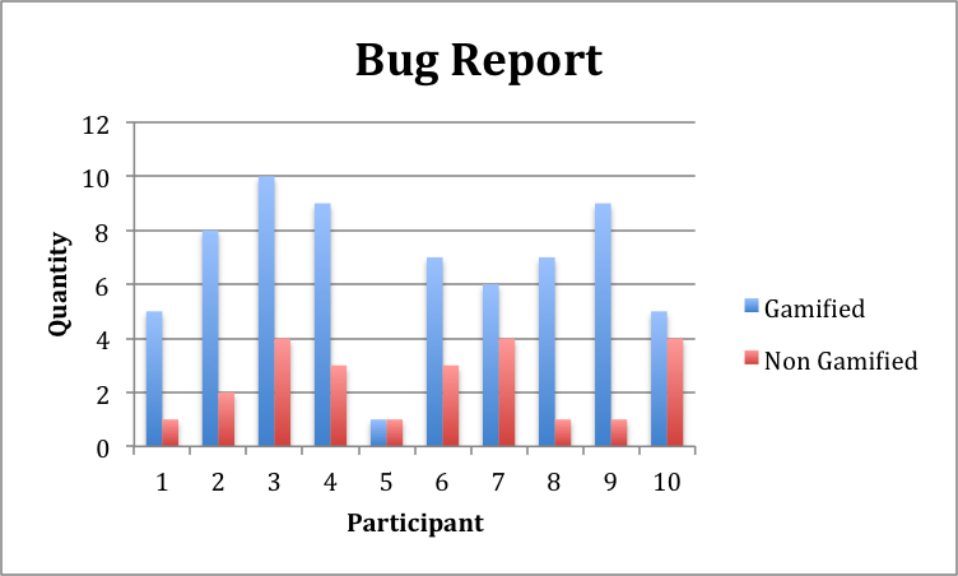


Figure A.34: Bugs reported by treatment and control group.

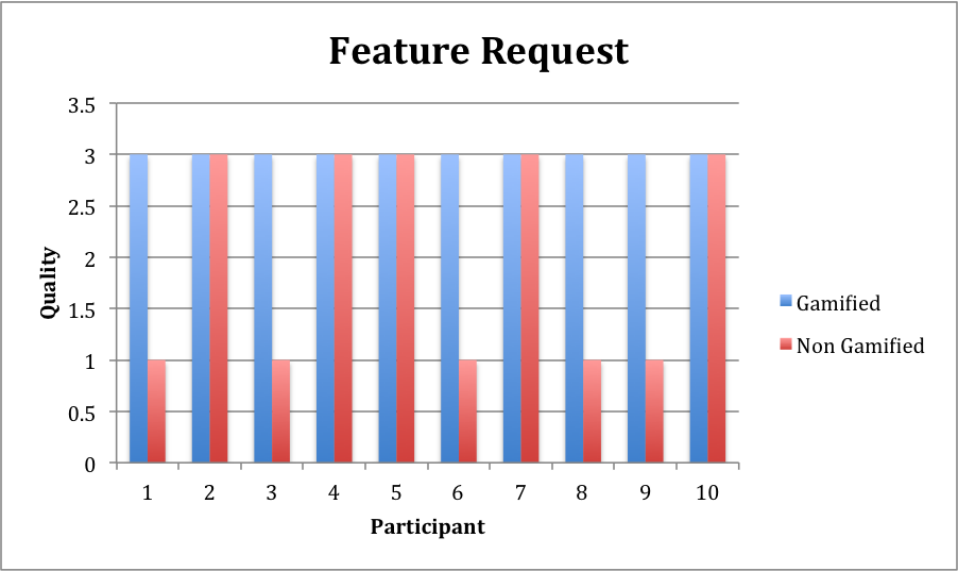


Figure A.35: Quality of Feature Requests reported by treatment and control group.

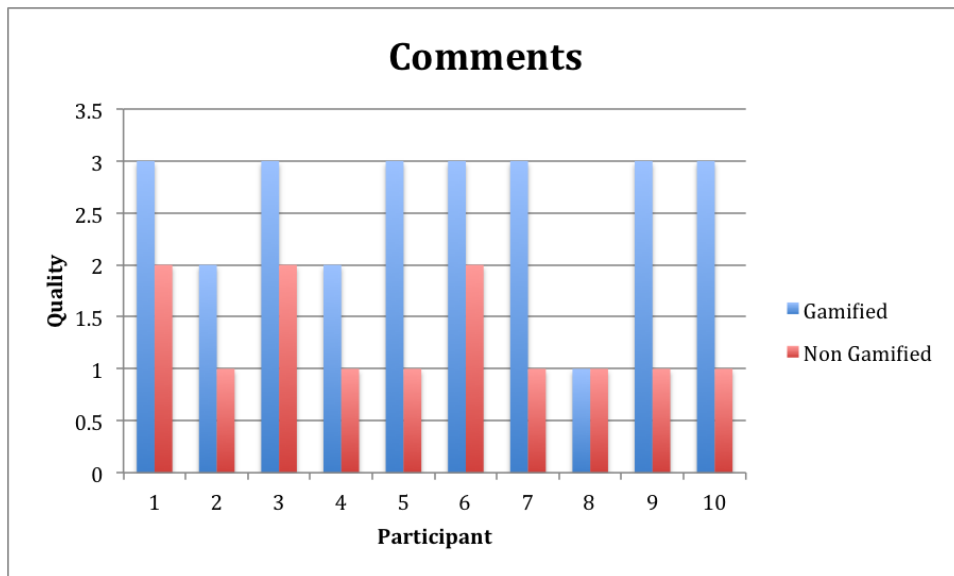


Figure A.36: Quality of Comments reported by treatment and control group.

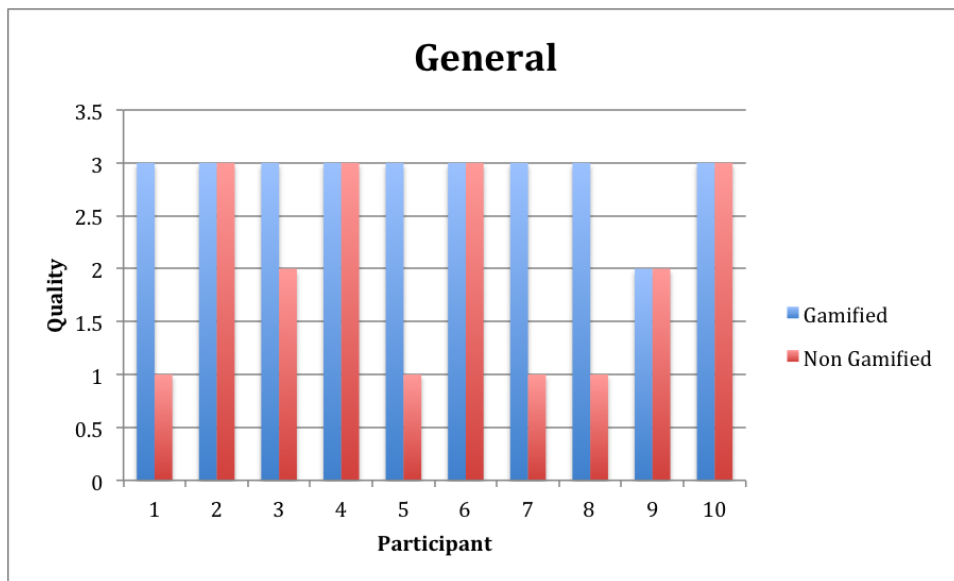


Figure A.37: Quality of General remarks reported by treatment and control group.

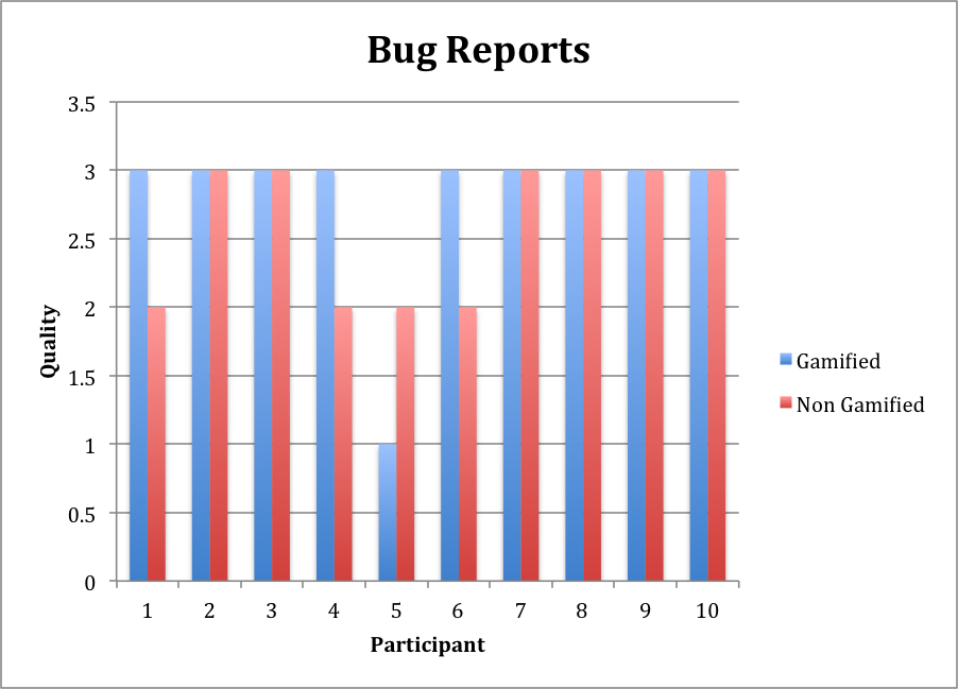


Figure A.38: Quality of Bugs reported by treatment and control group.

Bibliography

- [1] Belady L. A. and Lehman M. M. A model of large program development. *IBM Syst. J.*, 15(3):225–252, September 1976.
- [2] Bhattacharjee A. Social science research: principles, methods, and practices. 2012.
- [3] Canossa A. and Drachen A. Patterns of play: Play-personas in user-centred game development. In *DiGRA '09 - Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory*. Brunel University, September 2009.
- [4] Fisher R. A. Statistical methods for research workers. 1934.
- [5] Ko A.J., Lee M. J., Ferrari V., Ip S., and Tran C. A case study of post-deployment user feedback triage. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '11*, pages 1–8, New York, NY, USA, 2011. ACM.
- [6] Omoronyia I. Salehie M. Nuseibeh B. Ali R., Solis C. Social adaptation - when software gives users a voice. In *ENASE*, pages 75–84, 2012.
- [7] Kanstrup A.M. and Christiansen E. Selecting and evoking innovators: Combining democracy and creativity. In *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles, NordiCHI '06*, pages 321–330, New York, NY, USA, 2006. ACM.
- [8] Alain Ap. and Alain Ab. *Software maintenance management: evaluation and continuous improvement*, volume 67. John Wiley & Sons, 2012.
- [9] Burke B. Gartner says by 2015, more than 50 percent of organizations that manage innovation processes will gamify those processes. 2011.
- [10] Burke B. Gartner redefines gamification. 2014.
- [11] Brabham D. C. Crowdsourcing as a model for problem solving: An introduction and cases. *Convergence*, 14(1):75, 2008.
- [12] Eickhoff C., Harris C. G., De Vries A. P., and Srinivasan P. Quality through flow and immersion: Gamifying crowdsourced relevance assessments. In *35th Annual International ACM Conference on Research and Development in Information Retrieval (SIGIR)*, Portland, Oregon, USA, 2012.
- [13] Wohlin C., Runeson P., Höst M., Ohlsson M. C., Regnell B., and Wesslén A. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

- [14] Bacon D., Chen Y., Parkes D., and Rao M. A market-based approach to software evolution. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, OOPSLA '09, pages 973–980, New York, NY, USA, 2009. ACM.
- [15] Pagano D. and Brüggge B. User involvement in software evolution practice: A case study. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 953–962, Piscataway, NJ, USA, 2013. IEEE Press.
- [16] Carr H. H. and Dodd J. L. *Systems development led by end-users*, volume 45. 1994.
- [17] Ralph E. and William F. Refactoring and aggregation. In *In Object Technologies for Advanced Software, First JSSST International Symposium, volume 742 of Lecture Notes in Computer Science*, pages 264–278. Springer-Verlag, 1993.
- [18] Vouk A. Elmaghraby E., Baxter I. *An approach to the modeling and analysis of software production processes*. Addison-Wesley, 1995.
- [19] Gravetter F. and Wallnau L. *Essentials of statistics for the behavioral sciences*. Cengage Learning, 2013.
- [20] Jawaheer G., Szomszor Ma., and Kostkova P. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 47–51, New York, NY, USA, 2010. ACM.
- [21] Zichermann G. and Cunningham C. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. O'Reilly Media, Inc., 1st edition, 2011.
- [22] Sommerville I. *Software Engineering (9th Edition)*. Pearson Addison Wesley, 2010.
- [23] IEEE. *IEEE Standard for Software Maintenance, IEEE Std 1219-1998*, volume 2. IEEE Press, 1999.
- [24] Baroudi J. J., Olson M. H, and Ives B. An empirical study of the impact of user involvement on system usage and information satisfaction. *Communications of the ACM*, 29(3):232–238, 1986.
- [25] Brooke J. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
- [26] Dubois D. J. and Tamburrelli G. Understanding gamification mechanisms for software development. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 659–662. ACM, 2013.
- [27] Grudin J. Interactive systems: bridging the gaps between developers and users. *Computer*, 24(4):59–69, April 1991.
- [28] Grudin J. Systematic sources of suboptimal interface design in large product development organizations. *Hum.-Comput. Interact.*, 6(2):147–196, June 1991.
- [29] Koskinen J. Software maintenance costs. *Information Technology Research Institute, ELTIS-Project University of Jyväskylä*, 2003.

- [30] Schell J. *The Art of Game Design: A book of lenses*. CRC Press, 2014.
- [31] Bennett K. and Rajlich V. Software maintenance and evolution: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 73–87, New York, NY, USA, 2000. ACM.
- [32] Brennan K. et al. *A Guide to the Business Analysis Body of Knowledge*. Iiba, 2009.
- [33] Xu L. and Brinkkemper S. Concepts of product software. In *Eur J Inf Syst*, pages 531–541. Palgrave Macmillan Ltd, 2007.
- [34] B. P. Lientz and E. B. Swanson. *Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations*. Addison-Wesley, 1980.
- [35] Almaliki M., Nan J., Ali R., and Dalpiaz F. Gamified culture-aware feedback acquisition. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 624–625, Dec 2014.
- [36] DeBellis M. and Haapala C. User-centric software engineering. *IEEE Expert: Intelligent Systems and Their Applications*, 10(1):34–41, February 1995.
- [37] Fowler M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [38] Klein M., Moreno G., Parkes D., and Wallnau K. Designing for incentives: Better information sharing for better software engineering. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 195–200, New York, NY, USA, 2010. ACM.
- [39] Bartle R. Hearts, clubs, diamonds, spades: Players who suit muds. *Journal of MUD research*, 1(1):19, 1996.
- [40] Beyer H. R. and Holtzblatt K. Apprenticing with the customer. *Commun. ACM*, 38(5):45–52, May 1995.
- [41] Deterding S., Dixon D., Khaled R., and Nacke L. From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, pages 9–15, New York, NY, USA, 2011. ACM.
- [42] Deterding S., Sicart M., Nacke L., O'Hara K., and Dixon D. Gamification. using game-design elements in non-gaming contexts. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 2425–2428, New York, NY, USA, 2011. ACM.
- [43] Kujala S., Kauppinen M., Lehtola L., and Kojo T. The role of user involvement in requirements quality and project success. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 75–84. IEEE, 2005.
- [44] Zhiwei S. User involvement in system development process. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*, EASE '13, 2013.
- [45] Nosek J. T. and Palvia P. Software maintenance management: Changes in the last decade. *Journal of Software Maintenance*, 2(3):157–174, September 1990.

- [46] Maalej W. and Pagano D. On the socialness of software. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 864–871, Dec 2011.
- [47] Amatriain X., Pujol J. M., and Oliver N. I like it... i like it not: Evaluating user ratings noise in recommender systems. In *Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization: Formerly UM and AH, UMAP '09*, pages 247–258, Berlin, Heidelberg, 2009. Springer-Verlag.