Gamification of a Static Analysis Tool
A brief look into developer motivation

Version of May 26, 2019

Raies Saboerali

Gamification of a Static Analysis Tool
A brief look into developer motivation

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Raies Saboerali
born in Paramaribo, Suriname

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Gamification of a Static Analysis Tool A brief look into developer motivation

Author: Raies Saboerali
Student id: 4080335
Email: r.a.a.saboerali@student.tudelft.nl

Abstract

Software development is more than only implementing the functional code. A developer is also responsible for writing code measuring up to certain standards and conventions. These conventions make sure that the code is of a particular quality that improves readability and eases maintainability. Some of these conventions are checked by automated tools. Automated static analysis tools (ASATs) perform an analysis of the source code and issue warnings. ASATs are available for many programming languages and can be used to find functional or maintainability issues. Even though these tools have been proven to be useful during the code development process, developers do not always utilize them. The overload of warnings in large projects and relatively low importance of these warnings are one of the many reasons why they are ignored.

In this study, a gamification tool, Checkpoint, is developed based on the GOAL methodology. The purpose of this tool is to gamify the development process pertaining to ASATs to motivate developers. The developers are motivated using various gamification elements during a pretest-posttest pre-experimental experiment. The study tested the usability of the tool and its effectiveness. The experiment showed that gamification has an impact on developer motivation.

Thesis Committee:

Chair:	Dr. A.E. Zaidman, Faculty EEMCS, TU Delft
University supervisor:	Dr. A.E. Zaidman, Faculty EEMCS, TU Delft
Committee Member:	Dr. A. Katsifodimos, Faculty EEMCS, TU Delft
Committee Member:	Dr. M. Aniche, Faculty EEMCS, TU Delft

Preface

Starting a Masters degree was not planned when I first came to the Netherlands. During this journey, I have bee able to learn so much more about Computer Science. I owe my gratitude to Andy Zaidman who supervised during my Bachelors and Masters degree. Andy's guidance and feedback have always been helpful. During the last year of my study, there were some bumps, and I am grateful for Andy's support and understanding during this period. I would also like to thank Susanne van Ardenne for all her advice and help with study guidance and planning. The process of finding volunteers was not easy, thus I am grateful to Maaike Ruisendaal at Sogeti for helping me find volunteers for the experiment. I would also like to thank all the people who participated in the experiment of this thesis, and my friends who volunteered for beta testing.

Support from my family has played a huge part in moving forward and I am thankful for their support. I would like to thank my parents for their support and patience through all these years. Had they not worked so hard, I would not have never been able to study abroad. All of this would not have been possible without the two most important people in this journey; my uncle and my aunt. I am forever grateful to my uncle Fayek and aunt Sharida, without their support through all these years I would not stand where I am today.

Raies Saboerali
Delft, the Netherlands
May 26, 2019

Contents

List of Figures

Chapter 1

Introduction

Software engineering is more than coding, developers also need to comply with rules and regulations within their organization or team. Coding conventions are part of such rules. These conventions are guidelines for specific programming languages demanding certain practices and styles. Source code which violates coding standards has the potential to decrease the maintainability and readability of the software [38]. It is therefore of importance to follow these conventions. In order to keep the code quality high, developers use tools which assist them in finding bugs which have the potential to increase technical debt in the future. One of such kind of tools are Automated Static Analysis Tools (ASATs), which perform a static analysis on the code in order to find functional and maintenance issues [36]. Removing these warnings may make the software easier to maintain [3].

ASATs perform a static analysis of the code, in other words, the code does not need to be executed for the analysis. The goal of an ASAT is to find bugs and design issues [36]. Such a tool is used to assist developers in keeping the code quality of the software up to higher levels. The static analysis of the code can be performed on the non-compiled source code or on the bytecode [36]. FindBugs is one such tool which assists the developer in finding issues by analyzing the code at the byte level. Many other tools such as PMD and Checkstyle perform a similar analysis on the source code. ASATs can be used via the command line or as integrated tools within an integrated development environment.

ASATs are also used during code review, which is also a software development process focusing on quality control [26]. During code review, one or more developers inspect the code committed to the repository. The use of ASATs during this process reduces the time and effort of the review process [31].

1.0.1 Problems with ASATs

Even though ASATs have shown to be effective at finding design issues and contributing to code quality [38], there are still hurdles on this road.

Software Developers may find certain tasks tedious and boring [16]. Dealing with the warnings produced by static software analysis is one such task [16]. In order to make the best use of static analysis, developers use these tools in combination [5]. However, various tools give different warnings. Furthermore, their usage is still sporadic among many

opensource projects [5]. Very few projects mandate the use of ASATs and the removal of all warnings during development. Especially in larger projects, the probability of warnings is quite high. When using these ASATs, chances are that developers get overloaded with warnings from files they are not even working on. This creates a forest of warnings in which the developer may get lost. The tediousness of the task and the overload of warnings have a negative impact. Developers may start using ASATs irregularly during development or completely abandon the use of these tools. Based on these findings, it is safe to say that developers can be more productive when such tools are regularly used whilst writing code because they can see and fix the issues faster.

1.1 Research Question(s)

The amount of overload of warnings and the tedious nature of dealing with these tasks leads to developers ignoring the warnings [20]. The aim of this study is to find a way to make developers care about these warnings through gamification. Gamification is the use of gaming elements in a non-game context to improve user engagement and motivation [16]. Gamification has proven to be successful in motivating its users within software engineering environments [16]. Arai et al. also [2] conducted a study in which a gamified bug removal tool was developed. The tool was tested within a team of developers and resulted in increased bug fixes [2]. The purpose of applying gamification is not merely about rewarding users, but also to enrich the activities by enforcing a positive behavior [10].

The main goal of this study is to find out whether developers are motivated through gamification to resolve ASAT produced warnings. The study will also investigate why developers tend to ignore these warnings and what existing gamification models exist. As part of this study, a prototype gamification application will be implemented. The application will be put to test through a one-group pretest-posttest pre-experimental experiment. The gamification application will be named *Checkpoint*.

The research questions investigated in this thesis are:

- **RQ1:** Why do developers ignore warnings produced by ASATs?
- **RQ2:** What existing gamification models for improving software engineering processes exist?
- **RQ3:** Are developers willing to use gamification within their software development process in order to resolve ASAT produced warnings?
- **RQ4:** Does gamification encourage developers to resolve ASAT produced warnings?
  - **RQ4.1:** Do extrinsic motivators encourage developers to resolve ASAT produced warnings?
  - **RQ4.2:** Does competition through gamification encourage developers to resolve ASAT produced warnings?
  - **RQ4.3:** Which gamification mechanic is most effective in encouraging developers to resolve ASAT produced warnings?

- **RQ4.4:** Is Checkpoint effective in motivating developers to resolve ASAT produced warnings?
- **RQ4.5:** Is Checkpoint sufficiently usable for developers?

1.2 Thesis Structure

The study performed as reported through this thesis should be conducted with the application of appropriate methodology. In Chapter 2 the findings of the literature study performed as part of this thesis is presented. The findings reported in Chapter 2 will be used to answer RQ2. In this chapter, the attitude of the developer towards automated static analysis tools will be investigated. The background study will also report on the value of gamification within software engineering.

Chapter 3 will report the findings on existing gamification methods and its applications within software engineering. The study will show that applying gamification within SE is not simple. The gamification needs to be translated to characteristics of the particular domain. This translation will be executed through a proven methodology.

Based on the methodology the prototype, Checkpoint, will be developed. The design and implementation decisions will be presented in Chapter 4.

Chapter 5 details the experimental design. As stated, the goal of this thesis is to investigate the impact of gamification on developers resolving warnings. To properly investigate this, a proper experiment should be designed and executed.

The results gathered from the experiment will be presented in Chapter 6. In this chapter, the results from the pretest and the posttest will be presented.

In Chapter 7 the results will be interpreted in terms of the research questions. There will be a discussion on the dependent variables defined during the experiment. A deeper dive will be made into the impact of individual gamification mechanisms which have been implemented into Checkpoint. There will also be a discussion of threats to validity.

The study will be concluded in Chapter 8. In this chapter, the research questions will be answered based on the facts found through results and discussion. A brief overview of the contributions made by this study is provided. The thesis is concluded with possible ideas for the future in order to improve the current findings.

Chapter 2

Background and Related work

This chapter includes the related work performed on Automated Static Analysis Tools (ASATs) and gamification pertaining to software development.

2.1 Automated Static Analysis Tools

Automated static analysis tools are used to detect bugs or design issues. These tools assist the developer in finding issues which could lower the code quality. Various tools have been developed to audit the code. These tools detect issues by either analyzing the source code or bytecode.

FindBugs [1], PMD [2], CheckStyle [3] and JSLint [4] are a few examples of such tools. FindBugs, for example, is a well-known tool using bytecode analysis in order to audit the source code [21]. FindBugs has evolved under the new name, SpotBugs [5]. Over the years many tools have been developed for various programming languages.

2.1.1 Ignoring the Warnings

Many studies have been performed in order to understand how ASATs are used. ASATs throw a lot of warnings at the developer, one could expect that the developer would fix these warnings as ASATs are a tool for guarding code quality. However, often these warnings are not fixed [21]. In [29] they find that 56% of true warnings were acted upon by developers and 24% of warnings were false positives. Furthermore, a study at Google found that 55% of warnings were only acted upon after being entered into the bug tracking system [3]. Had these warnings been taken care of during the development, valuable time could have been saved into solving these issues. The study [3] found that the average lifespan of the bugs was between 5 and 9 months. Usually, older bugs are not fixed if they are marked as deprecated or obsolete [3]. Older code that does stay in the system tends to be more trusted

[1] http://findbugs.sourceforge.net/
[2] https://pmd.github.io/
[3] http://checkstyle.sourceforge.net/
[4] https://jslint.com/
[5] https://github.com/spotbugs/spotbugs

by other developers even if it would produce warnings by the ASATs [3]. If the code has been working correctly up till now, then there is no need to fix it and risk the chance of creating another bug. In [21] they find that only up to 9% of warnings are removed during the evolution of the source code.

The overload of warnings generated by ASATs may cause developers to avoid them [20]. Developers may not even be aware of the benefits of ASATs [20]. Another reason for ignoring these tools might be that the list of recommendations might be too extensive and can either be noisy or irrelevant to the developer [26]. Furthermore, not all warnings are important. In different projects, different types of warnings may be more relevant than others [26]. In [26] it is also shown that specific categories of warnings such as imports, type resolution, and regular expression were removed more often than others. Another concern is the high amount of false positives that ASATs produce [20, 29]. The lack of explanation of the warnings and effectively implementing quick fixes is also a disadvantage [20].

2.1.2 Practical value of ASATs

Static analysis tools are an affordable means of fault detection in Software. They are effective at finding design issues while developers can focus on more complex problems such as algorithmic errors [38]. However, the overwhelming amount of warnings, number of false positives and lack of explanation of the warnings have a negative impact on the use of ASATs.

An overwhelming amount of irrelevant warnings can cause the developers to lose sight of issues which need to be solved, or even demotivate them to get started.

2.2 Gamification

In this section, the value of gamification in software development will be explained. First, the argumentation will be made why gamification will be a useful addition to the development process. Afterwards, the gamification elements and modeling approach will be outlined.

2.2.1 Practical value of Gamification

Beller et al. [5] find the usage of ASATs to be common in popular OSS projects. ASATs tend to be used sporadically by developers and are not ubiquitous in popular OSS [5]. This can be caused by an overwhelming amount of warnings presented to the developer when using an ASAT. A significant amount of warnings indicate that the developer will need to put more effort into investigating.

Panichella et al. [26] found that specific categories of warnings were removed during the code review process, thus showing that static analysis tools also assist developers during the code review process in reducing the amount of warnings. The arguments in favor of ASATs suggest that these tools are a useful addition to the development process. However, software development is a human-centric and brain-intensive activity which requires motivation and discipline of developers [14]. Therefore creating a solution for motivating the use of ASATs

are of practical importance to the developer community and projected to be a beneficial addition to the process.

Software engineering tasks may get tedious at times. As argued before, the static analysis process is a valued one as it is used to audit the code for maintenance issues. Even though gamification has been a controversial topic [6], if implemented correctly, it will have a positive impact increasing user engagement and motivation. To model and implement an effective mechanism it is useful to understand why we play games, what elements attract us, and how to translate these into an SE (Software Engineering) process [10].

One popular example closer to the field of engineering is the well-known Q&A platform, StackOverflow. Gamification has been part of StackOverflow's success [35]. The platform awards its users with points for asking and answering questions. Each milestone is celebrated by awarding the user a badge which is publicly visible. These points and badges make up the user's reputation on the platform and are intended to further motivate them.

Gamification is also related to behaviorism since it is built on the concept of rewards [10]. The intuition behind behaviorism is not always trivial. Kohn found that removal of these reward elements may result in the user to return to baseline performance [10]. This does not mean that there is no hope. On the contrary, behaviorism is only a small part of gamification. McGonigal argues that the success of gamification originates from a reward which represents the foundation of optimal human experiences, such as satisfying work, the experience of being successful, social connection, and deeper meaning [10].

The purpose of applying gamification is not merely about rewarding users, but also to enrich the activities by enforcing a positive behavior [10].

Chapter 3

Gamification

The existing gamification methods and its applications will be explained in this chapter along with gamification within software engineering. Applying gamification in a software development process may not be as straightforward as it appears to be since it has to be transformed to the characteristics of the domain [14]. It is therefore of importance to study gamification within software engineering in order to design and develop an adequate system to fit within the development process of the daily programmer.

3.1 Behavior Change

The purpose of implementing gamification so developers resolve (more) ASAT warnings, is to make it a regular task. The aim is to create a new behavior such that developers who are not keen of resolving ASAT produced warnings, are motivated to resolve them on a regular basis.

Fogg presents the Fogg Behavior Model (FBM) [15] to understand the human behavior. In the FBM three important factors are presented: motivation, ability and triggers. To assert the target behavior to happen, it is siginificant for the developer to be sufficiently motivated, be able to perform the task, and be triggered. In Figure 3.1 the FBM [1] is visualized. Based on the studies performed, is it safe to conclude that developers are able to perform the tasks of resolving warnings, however, they are not motivated enough. In the application to be created, the motivation is of significance in this formula and vital for the success of the application's competence in persuading the developer.

Another interesting concept is the one of flow as introduced by Csikszentmihalyi [25]. In the article Concept of Flow[25], flow is defined as "a good life is one that is characterized by complete absorption in what one does". Flow is the optimal state in which a person becomes totally immersed in an activity [19]. When a person is in a state of flow, a great deal of enjoyment of the process is experienced and full of energized focus. However, when a task is perceived as too difficult, the person can experience frustration. On the other hand, if the task is too easy, the person can experience boredom. The trick when applying

[1] https://www.behaviormodel.org/

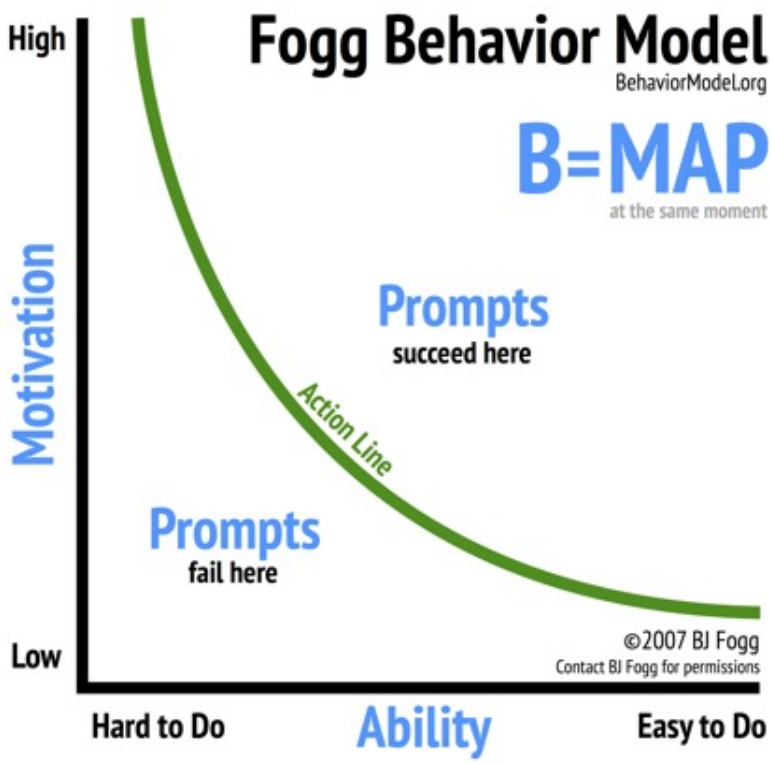

Figure 3.1: Fogg Behavior Model

gamification to any process, is to create an experience in such a way that the users spend their time in a state of flow.

In games this is done by leveling users up and increasing the difficulty as they gain experience. One of the challenges of gamifying the development process of resolving ASAT warnings is that many developers and teams have different levels of experience. In a video game for example, everyone starts at the beginning and progresses as they play. In gamification of software development, however, one team member may be significantly more experienced or familiar with the system than another member. Some teams may have never used ASATs and want to start using it. Others may have configured the tools but used them infrequently. These factors will be taken into account and used to delimit the scope of this project.

3.2 Motivational Drivers

In general there are two types of motivators, intrinsic and extrinsic motivators. Intrinsic motivation is linked to positive human behavior [34]. This type of motivator refers to autonomy, belonging, curiosity, learning, mastery and meaning. Extrinsic motivator refers to external motivators such as rewards, money, badges, points, competition, fear of fail-

ure, etc. Extrinsic motivators are effective for simple tasks is what Daniel Pink found [23]. When tasks get more difficult and require more effort, extrinsic motivators may turn into de-motivators [23]. The Self Determination Theory [30] suggest that extrinsic motivators may lead to motivated behavior when used properly [34]. Focusing on extrinsic motivators in such a way that leads developers to perform these tasks regularly is the key of this project.

3.3 Mechanics

Aside from intrinsic and extrinsic motivators, game mechanics play a significant role in the effectiveness of the gamification entity [1]. There are many mechanics with different levels of granularity and not all of them are particularly suitable in every business case. In this chapter the purpose and motivational driver of suitable mechanisms will be discussed. These mechanisms are points, badges, leaderboard, levels, and progress bars.

3.3.1 Points

Points are one of the most used gamification mechanics in the game world [27]. In many games some sort of pointsification system can be found. Pointsification is about rewarding the user's behavior with points [27]. Points are the most granular measurement in the gamification world, and they provide instant feedback to the user. Because of this, points contribute to the feedback as a motivation driver. Points may also be contributing to collecting as a motivation driver, because users see their points go up.

3.3.2 Badges

The second most used gamification mechanism is badges [27]. Badges represent the achievement a user has attained. If used correctly, badges may enforce positive behavior and address collection as the motivational driver. Badges can be awarded to users after completing tasks or gaining a certain amount of points.

Bunq for example is an online bank [2] which markets itself as a bank of the free. Customers of Bunq need to download an application in order to perform their banking operations. In their application they also make use of gamification in order to motivate their customers to perform a number of tasks. A slight difference here is that Bunq calls these trophies instead of badges. In Figure 3.2 the total collection of badges a user can achieve can be found. The design of this screen has been implemented in such a way that the user who is triggered by the motivational driver of collecting is compelled. Figure 3.3 shows the screen when a users has achieved a badge.

3.3.3 Leaderboard

Leaderboards are part of a social aspect of gamification. Leaderboards show a ranking of users in order to increase competitiveness among them. The position of the user on the board can be determined by many factors such, e.g. points, levels, etc. A leaderboard may also

[2] https://www.bunq.com/

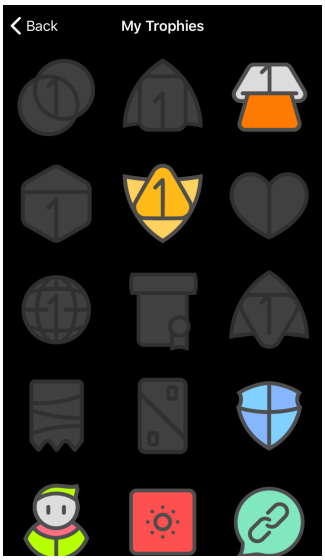

Figure 3.2: Trophies of Bunq

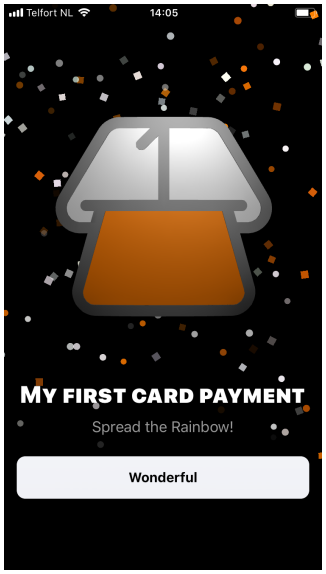

Figure 3.3: First Payment Award by Bunq

have a demotivating effect on the user, especially for new users who are starting with close to zero points while a top player might already have significantly more points. Players who become demotivated by this may not even try to play the game. It is therefore important to think about the consequences of this element and take appropriate design decisions in order to minimize the chance of demotivating users.

Some applications try to solve this by having cross-situational leaderboard [23]. In such a leaderboard the user is show in between other players who have more and less points respectively. Having a cross-situational leaderboard provides especially new users with a realistic starting point. Leaderboards may also be further divided among certain time periods such as weeks or months. Such leaderboards motivate the users to start over each period and provides the players with new opportunities to compete.

In Figure 3.4 an example of a leaderboard can be found. This leaderboard from Duolingo[3], a language learning app, shows by default the a weekly leaderboard between friends.

3.3.4 Levels

Levels are a powerful mechanism which address to the user's drive for achievement [12]. The levels mechanism is also used as a competitive mechanism in order to drive users in a social setting by showing their status. Levels are based on point rewards. As the user plays the game, points are earned, after which the user may get promoted if the threshold for the current level is exceeded.

[3] https://www.duolingo.com/

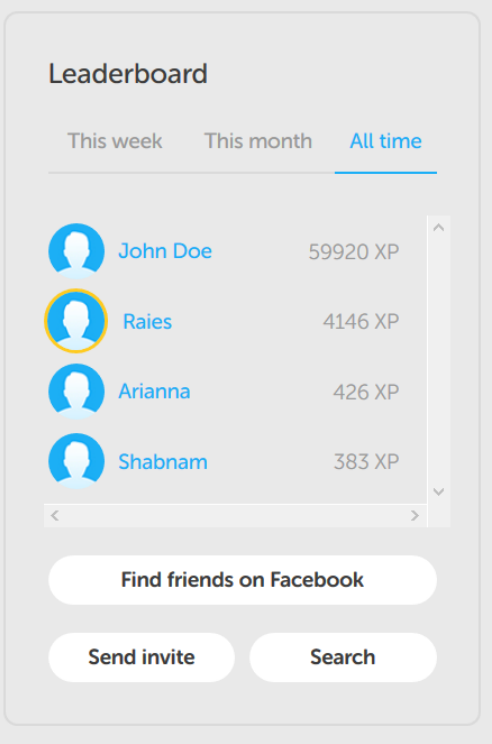

Figure 3.4: Leaderboard example of Duolingo

3.3.5 Progress

The progress mechanism provides the user feedback of the journey completion. Progress bars are used to display the overall progression and can be used to motivate users who are close to achieving their goal [22]. Progress bars are also used in combination with other gamification mechanisms. For example, by showing the progress of the user relative to the next level.

3.3.6 Game economy and rules

Game economy are the four basic currencies players can accumulate during the game [23]. The four economies are self-esteem, fun, social capital and fun. The game rules on the other hand, pull together the mechanics in order to create a flow which motivates the player. Game rules define when users are awarded points, level up, how much points they are awarded, etc. The purpose of the game should be taken into account when selecting the mechanics and designing the rules.

3.4 Gamification in Software Engineering

So far the various motivators and gamification mechanisms have been discussed. Although these subjects are important for our understanding of designing and implementing a gamified platform, applying these mechanisms to software engineering is not a straightforward task [14]. Garcia et al. argue that the essential difference between a game and software engineering is skill building enhanced by the influence of the gamification elements [16]. The research on applying gamification within SE, however, is scarce and premature, and provides no systematic methodology to incorporate these gamification elements within software engineering [16]. In order to support the integration of gamification within software engineering, the GOAL (Gamification focused On Application Lifecycle Management) methodology is proposed. The purpose of GOAL is to support integration of gamification in software engineering. Garcia et al. designed a generic software gamification architecture from which an engine has been built [16]. This engine has been tested in a software company. The blueprint for this project, GOAL, will be used in this project as a guideline.

3.5 The GOAL methodology applied

In this section the execution of activities of the GOAL methodology will be described. The steps one till four will be discussed for now. The last two steps, development and monitoring, will be part of the next chapter. First the objectives of the gamification will be identified, after which a player analysis will be performed. Thirdly, the gamification scope definition will be discussed. Based on the results of the data acquired during these activities, the fourth step, game analysis and design will be executed.

3.6 Defining the Mission

In Gamification at work, Janaki Kumar [23] states that the mission refers to the goal of the gamification activity. It is therefore vital to define a meaningful mission to guide the development process of the gamification tool.

The following steps are outlined in Gamification at Work [23] in order to create an effective mission:

- Understanding the current scenario. This is the first step in identifying the objective based on the results of the background study (Chapter 2).
- Understanding the target outcome
- Identifying a S.M.A.R.T. mission, which stands for specific, measurable, actionable, realistic, and time bound. Each letter corresponds to an objective, however, there are different sources referring different concepts [4].

[4] https://en.wikipedia.org/wiki/SMART_criteria

Current Scenario The work described in Chapter 2 has been performed in order understand the current scenario. Based on the findings, currently, a large portion of developers either use automated static analysis tools infrequently or they tend to ignore them.

Target Scenario The target scenario is the desired situation to move developers to. The target is therefore defined as the situation in which developers use ASATs more frequently.

The mission Based on the analysis of the current scenario and the target scenario, a mission needs to be defined. The mission needs to be specific, measurable, actionable, realistic, and time bound. The original mission as defined during this project was to get developers to resolve 20% more warnings in a fun and engaging manner within a span of two months. However, due to the nature of the experiment during this project the mission will be specific but not time bound. The M for measurable, which alternatively stands for motivating will be part of the mission. More details on the experiment setup will be presented in Chapter 5. The mission will thus be to get developers resolve warnings in a fun and engaging manner.

3.6.1 Player analysis

The next step is to perform a player analysis. However, since this project is not performed at a company, the player analysis will be based on purely fictional characters. It is still important to have some idea of the kinds of people which are going to use the gamification application, because the goal is to end up with a solution which is going to improve their engagement and motivation.

Based on the identified personas the organizations culture can be described as informal, cooperative, structured and group achievement oriented. The organization is also is relatively young in terms of average age of its employees. The players are motivated intrinsically based on their own duties and contribution to the projects they work on. The players are motivated extrinsically by being recognized for their work related accomplishments. Players may even receive a promotion at the end of the year for their performance.

3.6.2 Definition of Scope

This step is performed in order to define the scope of the gamification and to conduct a feasibility study, in order to choose the best solution.

Based on the objectives of the game and background study on ASATs, the area of code quality process with respect to ASATs will be gamified. Since it is not technically feasible within the scope of this study to gamify an ASAT plugin, an Ad-hoc solution will be applied in order to affect the process of developers resolving warnings generated by static analysis tools by gamification. The tooling will be web-based and enforce extrinsic motivators and it will be team-oriented.

3.6.3 Game Design

Based on the player analysis the following game mechanics has been selected:

Name	Jordan Peele
Gender	Male, 25
Job Title	Developer
Industry	Software Development
Job Goals	Combine technical expertise and business logic. Keep up with new technology
Pain Points	Needs to solve complex problems under time pressure
Aspirations	Be an expert at coding
Work Culture	Informal, cooperative, structured, group achievement
Player Type	Achiever

Table 3.1: Persona: Jordan

Name	Harold Kumar
Gender	Male, 36
Job Title	Senior Quality Engineer
Industry	Software Development
Job Goals	Become manager of the department
Pain Points	Difficulty dealing with new technologies
Aspirations	Become an expert at software quality
Work Culture	Formal, cooperative, structured, individual
Player Type	Explorer

Table 3.2: Persona: Harold

Name	Mia Hoffman
Gender	Female, 28
Job Title	Software Developer
Industry	Software Consulting and open-source development
Job Goals	Combine human experience within software technology
Pain Points	Has a busy schedule
Aspirations	Contribute to open-source projects
Work Culture	Informal, competitive, structured, group achievement
Player Type	Socializer

Table 3.3: Persona: Mia

- Experience: points the developer will be awarded when resolving warnings.
- Levels: after having accumulated a certain amount of points the developer will be promoted to the next level.
- Progress bar for level: the developer will be able to see how many points left to gain in order to be promoted to the next level.
- Progress over time showing how many warnings resolved: the developers will be shown how many warnings they resolve over time.
- Badges: a set of badges the developer can unlock by performing the tasks shown for each badge.
- Leaderboard: where the developers can compare the progress of each other by seeing where they rank.
- Activity log: where the developers can find a summarized timeline of their progress. This timeline will provide the developers continuous feedback on the progress they make.

The developers will obtain experience points as they play the game (resolve warnings). Gaining points will lead to leveling up, and probably increase their rank on the leaderboard. The game economics will be based on:

- Things: the use of experience points and badges.
- Self-esteem: the use of badges, level, and progress bars.

3.6.4 Changed game plan

During the first draft of the GOAL methodology, a streak count, was added to the game mechanics. Streak counts are also powerful gamification mechanics, especially if the number starts to get higher. Duolingo for example, shows a streak count for the number of consecutive days a player has achieved their daily goal. In this game the streak count would have shown how many consecutive workdays the user has committed code while resolving warnings. However, due to the nature of the experiment, this mechanism was dropped. There is no way the streak counter could have been tested fairly within an hour experiment.

In order to add a little bit of fun to the game, hidden achievements were going to be added as a mechanism. However, for hidden achievement to be effective the players of a game need to get familiar by playing the game for a lot more than just an hour. Hidden achievements are not always easy to unlock, they require some experience of the game, which is exactly what makes hidden achievements interesting.

Instead of the hidden achievements, it was decided to add a virtual buddy to the game in order to add a little bit of fun. The virtual buddy would change its expression based on the activities the user performed.

Chapter 4

Design and Implementation

The previous chapters have discussed about gamification and reason why ASAT produced warnings are low prioritized or even ignored. The purpose of this chapter is to show how these results led to the implementation of the tool in this project. Screenshots of the implemented Checkpoint can be found in Appendix E.

4.1 Ad-hoc or Integrated solution

On of the early decisions during the design and implementation process was deciding to make the tool integrated or ad-hoc. In Chapter 2 it is reported that the gamified tool needs to be incorporated into the software engineering process. The choice between these two options would impact the whole development process and design of the tool.

An integrated tool, in this case, meant developing a tool integrated into the IDE. The plugin would be running while the developer was coding in the IDE. During the development, the tool would analyze the modified code. An ad-hoc solution meant a separate tool which would analyze the code.

An integrated tool could work offline. The tool could analyze the code each time the developer saved modified or new code. Such a tool would also require a connection with a server to synchronize the results of the developer and teammates. However, this design would make cheating relatively easier. Because everything is processed offline, the developer could easily cheat the system. This could be achieved by for example creating warnings and resolving them on your own. Other developers would not even be aware about this behavior. Another downside to this solution is, one would have to develop a tool for every IDE. Or at least, every major IDE.

The ad-hoc solution, on the other hand, would be a tool running on a server. Developers could view their progress through a web portal. To facilitate this, the tool would require access to the repository. Each commit is analyzed and based on the results the gamification engine would act accordingly. This system makes it harder to cheat because everything is processed remotely. A developer could still deliberately introduce warnings to resolve these and gain points. However, these commits would be shown in the history of the developer and could possibly hurt their reputation. An ad-hoc solution would be required to read the

repository and commits.

Based on this analysis, an ad-hoc solution was chosen. The tool was named: Checkpoint.

4.2 Requirements

Before implementing the actual tool, a set of requirement were setup in order to guide the implementation. The requirements were setup based on the information from the previous chapter. The requirements are as follows:

- Warnings should be scanned and logged.
- Based on warnings resolved, the user should be awarded points.
- Accumulating a number of points should award the user with badges.
- Resolving a specific number of warnings should award the user a badge. The following combinations are possible:
  - x warnings of type Y
  - x warnings of category type Y
- Accumulating a number of points should level the user up.
- A global leaderboard should be available where the positions of the player is displayed among other players.
- The user should be displayed an activity log where the progress is displayed. E.g. which achievement is unlocked.

The application about to be designed needs to comply with the requirements above.
The following are the technical requirements for the application:

- MySQL Database
- PHP 7.1
- CakePHP 3.6 as a MVC framework
- firebase/php-jwt for JSon Web Token generation
- Checkstyle v8.11
- Java v8
- Git 2.14
- Smee.io
- GitHub App

4.3 Designing the tool

When starting to design, the tool was visioned to run on a server with a web portal for every user. The server would listen to events of remote repositories. The web portal would contain the gamification elements. These ideas led to the design of the system. The back-end of the application consists of an event engine, analysis engine, and game engine. In Figure 4.1 a high level overview of the design can be found. When the developer pushes a commit, the the changes are received by the remote repository. The repository sends an event to the server of Checkpoint. Checkpoint will be configured to listen to webhook events of the remote repository. The developer, can log-in on a web-portal and view the results. The results are retrieved from the server.

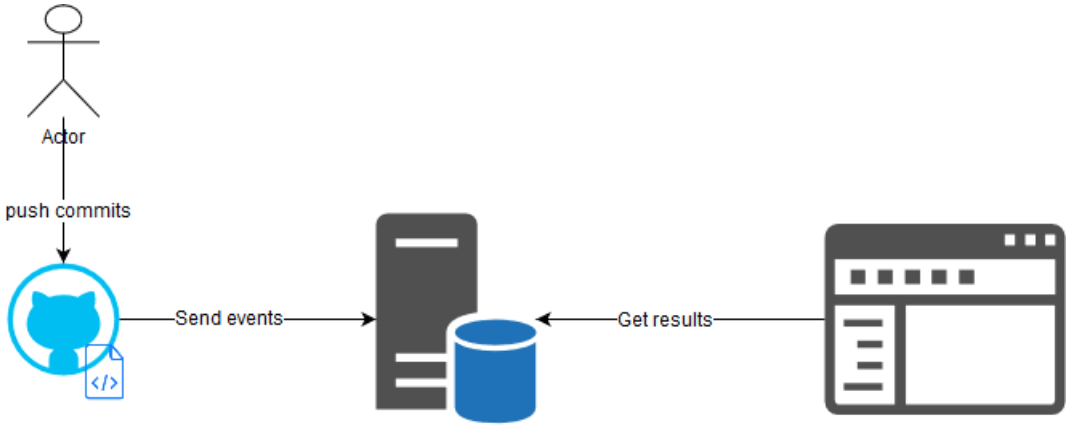

Figure 4.1: Overview of Process

4.3.1 Event Engine

The event engine is responsible for handling requests from the remote repository. For this project, GitHub was chosen. GitHub is a widely used hosting service which uses Git as its version control. The tasks of the event engine are to process the webhook events and log the changes of the commit to the database. GitHub is capable for sending out many types of events [1]. The events are received in JSON format. The significant events for Checkpoint are *push*, *repository*, and *installation*. The *installation* event is fired when a new GitHub app is installed or its permissions are altered on an existing installation. The *repository* event is fired when a repository is created, deleted, archived, un-archived or edited.

After installation of a new repository, the server receives the event. The event data is processed, and the an entity for the project is saved in the database. This process also checks if the project already exists in the database. The reason for this is failed webhook events. If an event fails to be processed midway, the event should be processed again without errors. For each project, a folder on the server is created. In this folder *Git* is used to clone the

[1] `https://developer.github.com/webhooks/`

entire repository. After cloning, the project is analyzed. The same process is executed for the *repository* event. The main difference for this process is, an *installation* event may contain multiple repositories, because the GitHub app can be installed on an organization with multiple repositories. A *repository* event pertains only a single repository.

This process may take a while, because all the files in the project need to be scanned using Checkstyle. For *push* events, the JSON data contains and array of commit information. The JSON contains the hash, author, commiter, files modified, files added, files removed, and some other data. When a *push* event is received, the commit data is saved into the database. During processing, the author in the commit is searched for in the database. If the author is registered in Checkpoint, a link is made between the commit and the author. For each file, the same actions are executed. Each file in the JSON commit data is saved to the database and linked with the registered author.

4.3.2 Analysis Engine

The analysis engine has the responsibility of processing and analyzing every commit which has been processed by the Event Engine. Processing for the Analysis Engine means running Checkstyle and computing solved warnings for each commit.

The event which is going to be most used by the program is the *push* event from GitHub. Whenever this event is received, the commits are saved with their relevant details. Afterward, Checkstyle is executed on the server given the file to check. Checkstyle on the server was configured to run using *sun_checks*. This is one of the two default configurations within Checkstyle. The configuration was also set to default within the IDE used during the experiments. Given the purpose of this project, it was not deemed necessary to implement reading custom Checkstyle configuration for each project. *sun_checks* also resulted in more warnings than the *google_checks* configuration. This benefited the experiment because with more warnings it would be easier to stimulate annoyance of warning overload. Checkstyle version 8.11 was used in the IDE and on the server to check the files. The tool was executed using *shell_exec* and the results were formatted in XML. The XML was then parsed and saved as an entity in the database. Each warning entity was linked with the file, commit, and project. During this step, each warning entity was also linked with a warning type entity. This was performed by using the source error parameter of the Checkstyle results and check it against known warning types. The classification of warning types will be explained in Section 4.5; The warning type is an important link here, because it is used for serving points and keeping track of types of warnings resolved.

Checkstyle is executed on modified and new files. A list of these files can be found within the event data from the webhook. At the end of this process, the commits are marked for being logged for warnings. This step of the process is also developed defensively to avoid duplicate warnings during a failure during processing. Each warning for a certain file can only be saved once in the system. The check for duplicates is performed using file, line, column and the warning itself.

After the warnings have been saved to the database, the next step is to compute the resolved warnings. This step is executed for each commit in the *push* event. For each commit, the revision is checked out on the server. Using Git the commit hash history of

the file is extracted. Git is executed by using a the *exec* command of PHP. If the command is executed with success the revision before the current one is returned. The hash of the commit is used to get the commit entity in the database. Previous warnings of the file are retrieved from the database using the identified commit entity and file name. The *current warnings* and *previous warnings* are then used to compute the warnings resolved. Each resolved warning is then saved as a warning action entity in the database. The warning action entity keeps track of each warning and the commit in which it is resolved. This step will be explained in more detail in Section 4.4. For now, it is important to remember that warning action states which warning is resolved in a commit, and how much XP the action is worth. After resolved warnings have been computed, the Game Engine it is the turn of the Game Engine to process the warnings.

If a *repository* or *installation* event is received, Checkstyle is executed using the same procedure to analyze the files. The step to compute warnings resolved is skipped during these events. At first, Checkstyle was run on every file during initialization of the repository. This turned out to be an exhausting operation. It took nearly thirty minutes on average to process the repository used during the experiment. This step was optimized by only executing on Java files. By doing so, the initial scan was cut down to an average of five minutes.

4.3.3 Game Engine

The responsibility of the Game Engine is to compute the changes relating to game elements based on warnings resolved. To achieve this, warning actions are used. After the analysis is complete, each commit during the *push* event should have warning actions. If there are no actions present, then no warning is resolved. The achievements (Section 4.6) have a value and a type. An achievement can be either a badge of just points. Achievements have tasks.

During this process, all warning actions of the commit are taken into consideration. The steps are as follows:

- Get warning actions of the commit.
- Increase the warnings resolved counter for each task.
- Compute the XP gain for each unprocessed warning action in this commit.
- Update the XP of the user.
- Increase the points tasks on the user based on the XP gained.
- Unlock the applicable achievements.

In the second step for each warning task, the resolved warning counter is incremented if the warning action is applicable to the task. The warning action is only applicable to the task if the warning type of warning action and task are a match. For example, a task demands to resolve Java doc warnings for methods, and there is a warning action found in the actions of this commit representing a resolved Java doc warning. Then the warning

resolved counter for this task is increased. If the tasks appear to be completed by this action, the task is marked as completed for the user. All warning actions are compared against all the uncompleted warning tasks to find a match and increase counters.

In the third step, the total XP gain from all unprocessed warning actions in the commit is computed. The total XP gain is then added to the total XP of the user. The same XP is also used to increase the points for all the point tasks for the user which are not completed yet.

After these steps, the warning actions are processed and counter for all tasks are updated. The achievement component will then check which achievement the user deserves. All the locked achievements of the user are retrieved from the database. For each achievement, all the tasks are checked for completion. If all tasks are marked as completed and the achievement is locked, then the achievement is handed out to the user. If the achievement is worth XP points, the XP is also added to the total XP of the user. After handing out achievements, notifications are created in order to inform the user. The changes are also written to the activity log. The activity log is visible on the right side of the screen.

After each XP update of a user, events are triggered to invoke relevant updates. First, the level of the user is checked. If the new XP of the users applies for a level upgrade, the user's level is updated. The total points of the user on the global leaderboard is also updated. Other users are also able to observe this change. The commit entity and warning actions are used to display the warnings resolved graph on the dashboard.

4.4 Warnings Resolved in Action

To hand out achievement it is crucial to keep track of warnings resolved. The algorithm for determining resolved warnings was one of the key decisions in the process of developing this system. Tools like Checkstyle, are as they are called, static. In order words, the tools themselves do not keep track of warnings which have been resolved. Checkstyle only scans the source code and shows the warnings based on the provided configuration. It is also not Checkstyle's purpose to keep track of resolved warnings. However, for this system it is important to keep track of resolved warnings. Resolved warnings are the one thing which needs to be measured for for the game mechanics to work.

The way resolved warnings are computed has an impact on the user experience of the system. If the system discards a noticeable amount of the warnings a user has really resolved, users may lose faith in the system. On the other end of the spectrum, the gamification may become too easy if more warnings are marked as resolved than the user actually resolved.

To decide how the resolved warnings would be computed, diffs from personal projects and random Java projects on GitHub were investigated. This resulted in the idea to investigate code changes.

The first option was to track changed lines of code. Canfora et al. [9] proposed an algorithm for identifying changed source code lines.

The second option was to track code regions. Duala et al. [13] proposed a technique to track code clones in evolving software. Code clones are generally considered harmful

[13]. Duala et al. also proposed a code clone management system based on Clone Region Descriptors (CRDs). The CRDs were also improved to track code when moved to other locations by Higo et al. [18]. This seemed interesting, if the moved code can be tracked then, the warnings could also be possibly tracked even when moved. The tracking works by parsing the source code into an abstract syntax tree and approximating the distance between the blocks identified.

The first option explores the idea of tracking changed lines. If this option were used, warnings resolved would be computed based on changed lines. For each line that was changed, the previous version of the file would be checked in which the same line was present. However, tracking changed lines is not perfect. The second option is a broader approach to this issue. Instead of looking at lines, code regions are compared. The algorithm for the second approach was not provided, this was one limiting factor for implementing this algorithm. The accuracy of the algorithm depends on its implementation.

Both of these options have one important issue. They limit warnings to specific regions. The problem is, at this point, there is no knowing how the users are going to view the resolved warnings when used in Gamification. Not to forget, the main goal of this experiment is to find out the effect of gamification on the process of resolving ASAT produced warnings.

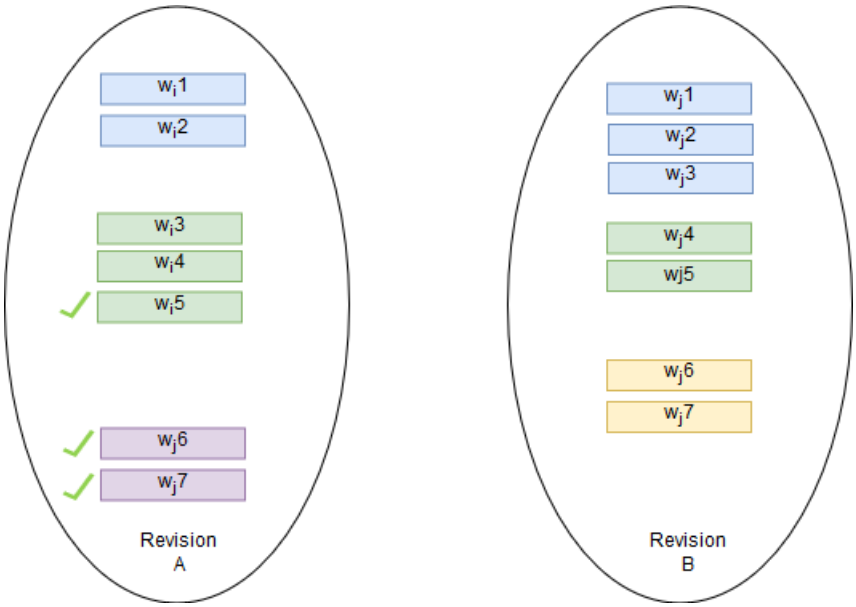

Figure 4.2: Warning Set Example for a File

A third option was considered. Instead of identifying changed lines or code regions, look into warnings on file level. For each file, a number of warnings are produced. This approach would then take into account all the warnings of the current version of a file and compare the warnings to all warnings in the previous version of the file. The warnings would be compared based on their type. The algorithm would perform a count for each

warning type in both versions of the file. This would result in two sets of warnings types and a counter. These two sets would then be compared by subtracting the old set from the new set of warnings resolved. The difference between these two sets would be the result of resolved warnings. During the processing of a *push* event, the warning action would be created by marking a warning from the old set as resolved.

In Figure 4.2 an example for one file is visualized. This example contains a set of warning for some file of revision A and revision B, with revision B being the newer version. The colors on this set represent the warning types. In this example one can see that of warning type blue, no warnings are marked as resolved. This is because, in revision B, more warnings of type blue exists. One warning of type green is marked as resolved because the newer revision contains one less green warning. The purple warnings are all marked as resolved because they are not present in revision B. The green warnings are completely new, and therefore no further actions are taken against them.

4.5 Warning Classification

To hand out points for the warnings resolved, each warning will need to be assigned a reward value. Checkstyle version 8.11 has 154 warning types. Each type of warning is different, some types may be perceived as more difficult or more work to fix than other warning types. For example, a white space warning is easier to fix than missing Java doc method documentation or resolving magic numbers.

Classifying 154 types of warnings would not be easy. The General Defect Classification [2] provides a way to classify warnings from several tools under mutually shared categories [7]. The GitHub repository for the GDC already contained a list of Checkstyle warnings being classified under the GDC. This list was compiled for version 6.6 of Checkstyle at the time this repository was accessed. New types of warnings had to be manually classified. The classification was performed by finding the best fit of each Checkstyle warning within the GDC based on the description of a GDC category and the warning type. All 154 warnings were classified within the 18 GDC categories. The GDC divides these categories within functional and maintainability warnings. This distinction was used when assigning points.

Each warning type was linked with a warning category within the system. Each of the GDC warning categories was assigned a reward value. In Figure 4.3 the link between warning type and warning category is visualized by showing part of the database schema. Functional category warnings were assigned more points than maintainability warnings. Although the system allows for assigning a reward value for each warning type, rewards were only assigned to categories. Assigning values to each warning type would have been redundant in this case.

4.6 Achievements

The achievements are an integral part of the system representing the link between tasks and users. A simple and abstract depiction can be found in Figure 4.5. The green tasks represent

[2] `https://github.com/Inventitech/gdc`

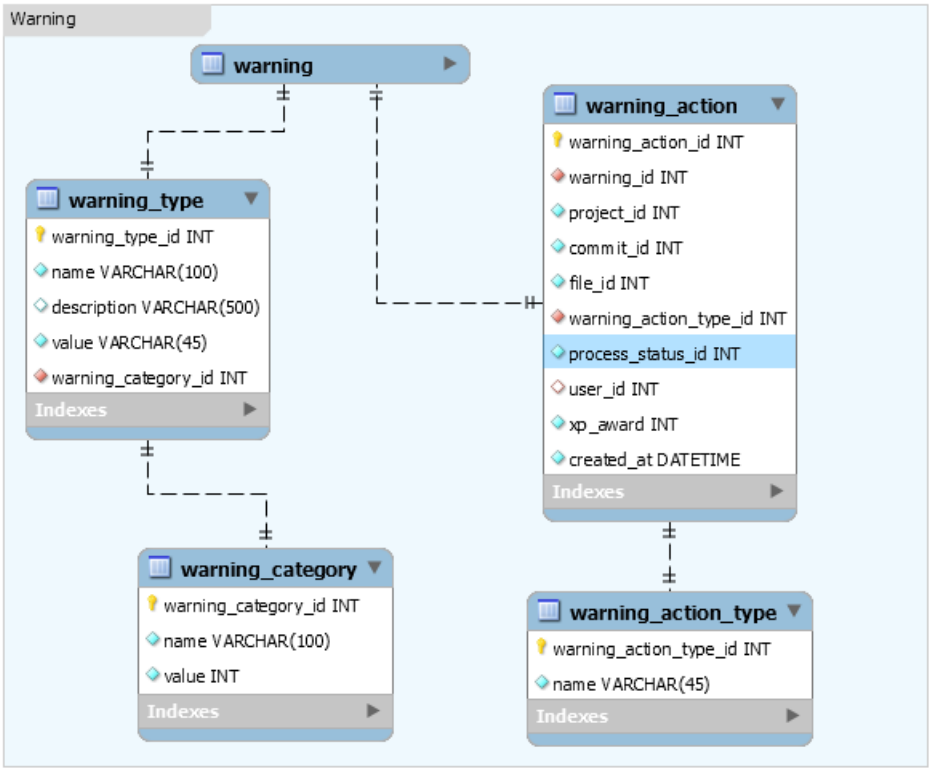

Figure 4.3: Warning Schema

completed tasks, and blue achievements represent unlocked achievements. Achievement A2 in this example in not unlocked, because not all the tasks for achievement have been completed. Achievement A3 on the other is unlocked, because all the linked tasks are completed.

Part of the database schema representing the Achievements are visualized in Figure 4.4 . At the top of the schema, the level, achievement, leaderboard and user activity can be found. The achievement element is the largest in terms of design and implementation. The achievement represents a badge achievement or points achievement. Distinctions between the two types are achieved by the achievement type.

4.6.1 Tasks

Each achievement has a value, rewards points which are awarded when unlocked. Each achievement has tasks. The tasks are divided into warning tasks and point tasks. Each task has an evaluation mode and activation mode. The evaluation mode represents when the task should be evaluated. The idea was to create a distinction between tasks for every commit and tasks which should be evaluated monthly or weekly. This part is therefore not relevant anymore, because of the setup of the experiment. The activation mode and completion value

provide the formula for task completion. The only activation mode used in this experiment is ¿=, therefore rendering this part irrelevant for the experiment.

All tasks can be coupled to multiple achievements. Point tasks are tasks which activated when a certain amount of points are earned. Warning tasks represent the number of warnings of a certain type which should be resolved. The completion value represents the number of warnings to resolve. The warning type or warning category is used to determine which warning should be resolved.

Upon registration, links are created between the user and all the tasks and achievements. These diagrams can be found in the lower part of Figure 4.4. For each task, a value is kept track of. The value represents the points acquired or warnings resolved for point tasks and warning task.

4.6.2 Unlocking

The user achievement keeps track of the achievement unlocked by the user. When processing a commit, the warning actions are compared against the tasks. The value for each user is then incremented and stored if either the warning type of warning category match with the warning resolved in the warning action. If the value of the task is equal or greater than the completed value, the task is marked a completed. User achievement is subsequently used to get all the unlocked achievements. For all these unlocked achievements, the links between all the tasks for the user is audited. If all the tasks are completed, the achievement is marked as achieved.

4.7 Git and GitHub

The Git component and GitHub component within Checkpoint are crucial elements. The GitHub component is not only responsible for handling webhook events from GitHub, but also in making a secure connection from the server to GitHub.

After an event is received and a new repository needs to be installed, or changes need to be pulled, the server makes a connection with GitHub. In order to be able to successfully pull those changes, the GitHub App has to be authorized [3]. For this authorization, a JSON web token is generated on the server. The generated JWT token is used to make a call to the API of the installed repository. If the call is successful, GitHub's API will return a token. This token is then used by Git on the server for operations which require authentication. An overview of this process can be found in Figure 4.6. A *git pull* operation required such a token in order to fetch changes.

A GitHub App was chosen instead of an OAuth App. The decision was made based on the information provided by GitHub [4]. OAuth Apps can only be installed per user on a repository, while a GitHub App is installed for the user or organization and has access to all repositories. The OAuth App acts as a user, whereas the GitHub App is installed for

[3] https://developer.github.com/apps/building-github-apps/authenticating-with-github-apps/

[4] https://developer.github.com/apps/differences-between-apps/

everyone working on the repository using its own identity. This is a major difference for Checkpoint. The GitHub App makes teamwork easier because the App is installed as an independent entity watching over all changes made by each developer. Furthermore, in an OAuth App, the user has to configure the webhook URL on their own. In the GitHub App, the webhook URL is configured within the App.

4.8 Webhook Routing

Ideally, an application running on a web server would receive webhook events from GitHub directly. However, for this experiment, the application was run on a local server. Webhook events cannot be sent to localhost. In order to redirect the connection, the GitHub App was rerouted. The overview is depicted in Figure 4.7.

Smee was used for handling the webhook events. Smee is a webhook payload delivery system [5]. A smee channel can be created online or locally after having installed the smee client. The smee client should be installed on every machine which wished to listen to smee. The GitHub App, in this case, was configured to send its payloads to the smee channel. Smee, when running locally on the experimenter's computer, would then listen to all the events. The events would then be routed to a local webhook URL.

[5] `https://smee.io/`

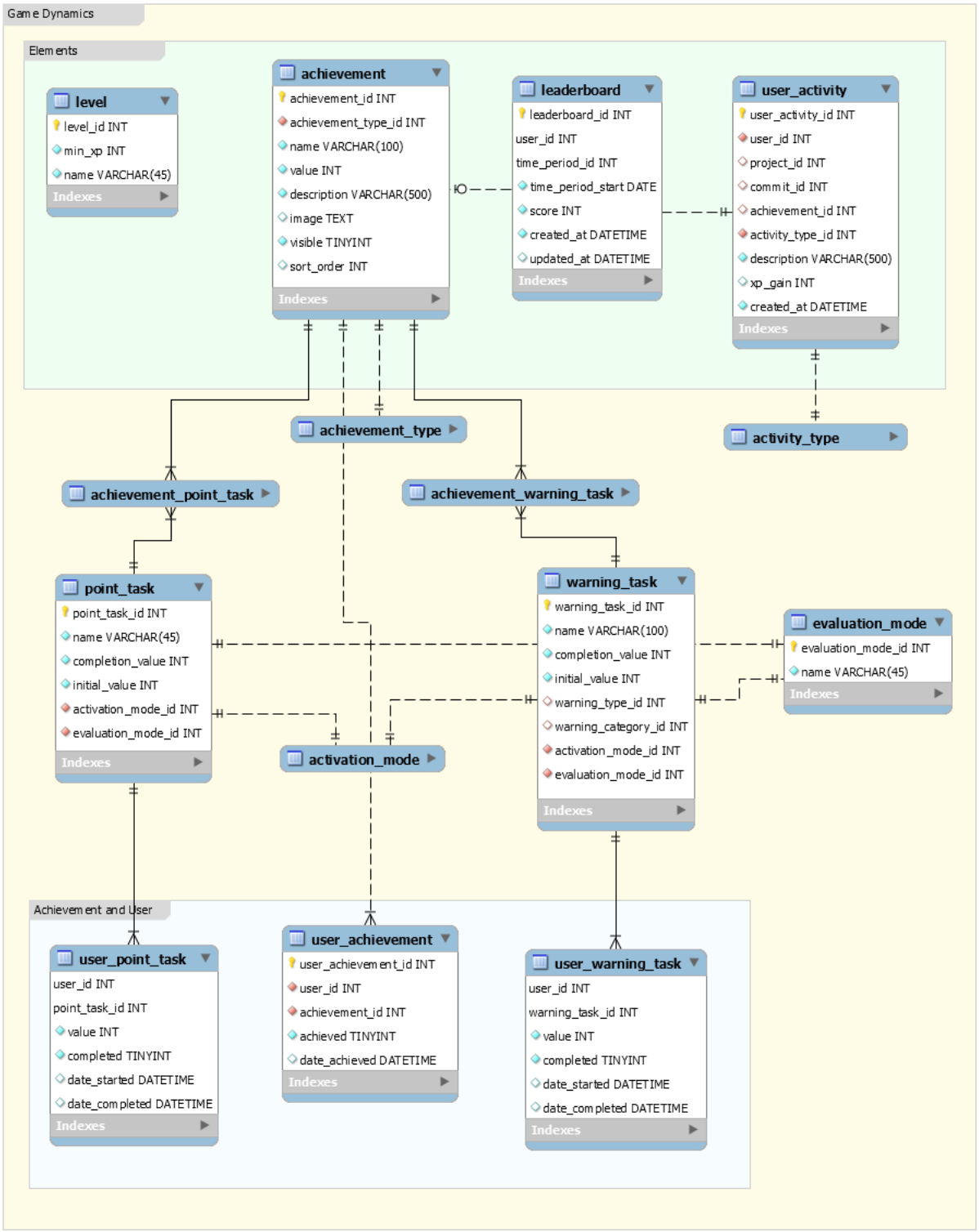

Figure 4.4: Achievement Schema

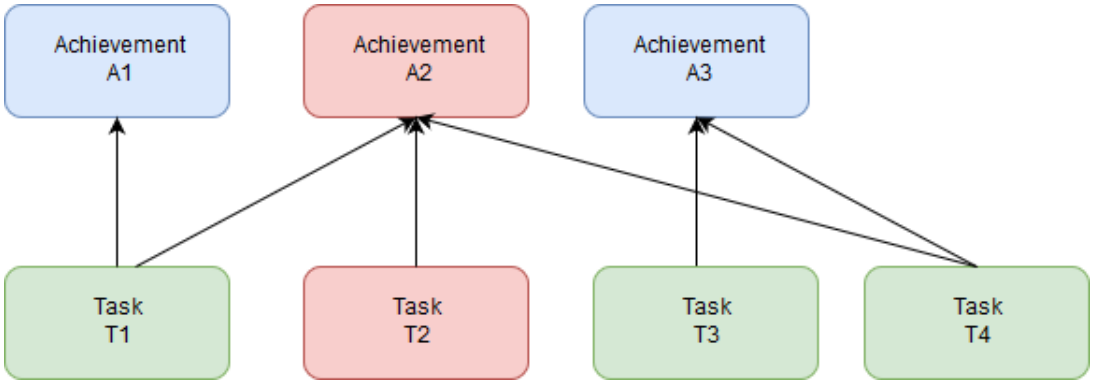

Figure 4.5: Achievement and Tasks

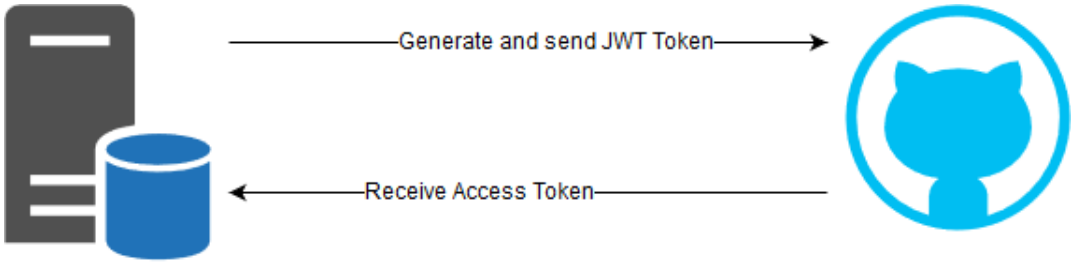

Figure 4.6: GitHub App Authentication

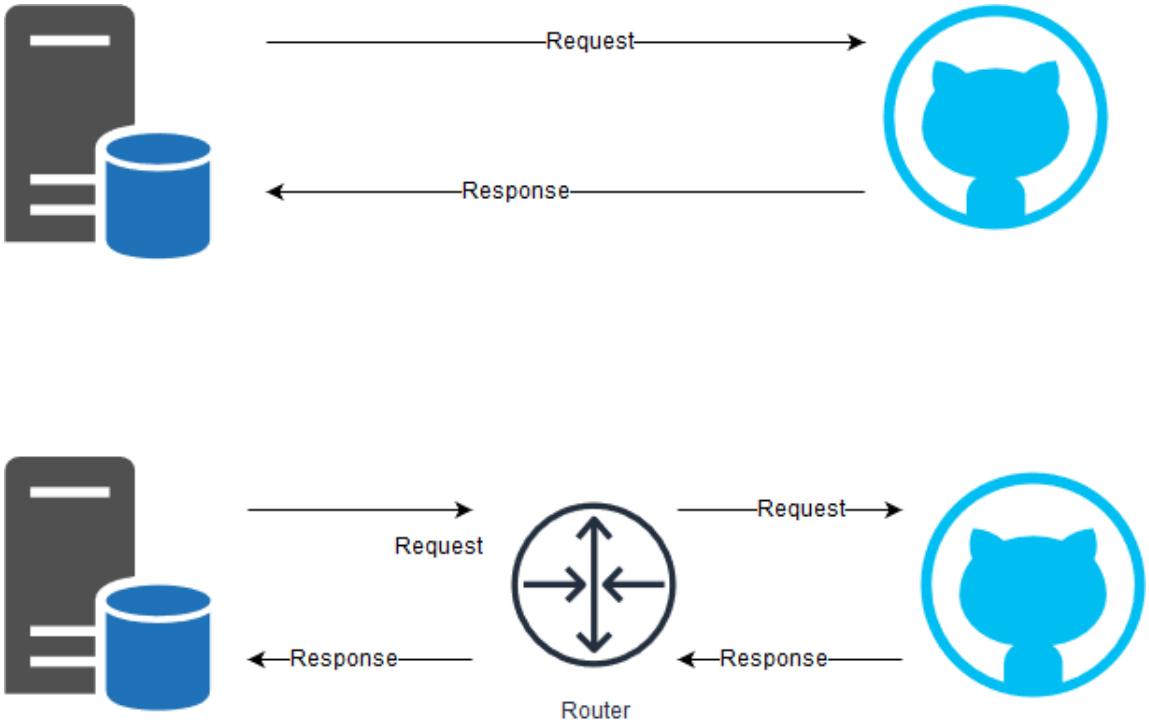

Figure 4.7: GitHub App Connection

Chapter 5

Experiment

In order to answer the research questions the tool developed as part of this project will be evaluated. The goal of this experiment is measuring the effectiveness of Checkpoint and impact of gamification on the software development process. A number of subjects were selected and tasked to perform a number of programming assignments within Eclipse while using Checkstyle and Checkpoint. The development environment was set up beforehand, the participants were asked to focus on the assignment itself. For the experiment type, a pretest-posttest pre-experimental design [4] was chosen. An interview was conducted at the end of the experiment. The purpose of the interview was to gather insights from the participants which would not have been possible with a questionnaire.

In this chapter the design of the experiment will be explained. This will be followed by the pretest-posttest and interview setup. The chapter will end with the setup of the experiment and the execution description.

5.1 Design of the Experiment

The experimental design is an important part of an experiment. A good experimental design will allow the researcher to gather credible results and draw sound conclusions. The experimental design and proper documentation are also essential for the repeatability of the study. It is important to distinguish which variables are to be tested and which approach is best suited to investigate the impact of these variables. In a scientific experiment, two main variables, the dependent and independent variables are tested. The independent variable is the variable which is either being altered or controlled in order to measure the effect on the independent variables [17]. The dependent variable is the one being tested. These variables may also be seen in terms of cause and effect [17].

5.1.1 Variables

The primary subject of this experiment is Checkpoint, a tool developed as part of this experiment. Checkpoint has be implemented based on the gamification mechanisms as explained in Chapter 3 with the aim to gamify part of the development process. Checkpoint will be

the independent variable in this process in order to measure the effect of the application on the development process.

The effect of Checkpoint is what is being measured. A good starting point in defining the dependent variables are the research questions which have been defined in Chapter 1. The first and second research question are related to the background and technical issue respectively.

The last research question is addressing the effect of the gamification onto the process. In this research question there are four sub questions. Each of these questions address a different issue. The first question focuses on the effect of extrinsic motivators while the third question focuses on the effectiveness of the gamification mechanics. The dependent variable from these questions map to the ability to motivate. The second question focuses on the competitiveness. This maps to competition as a dependent variable. The fourth and fifth questions map to effectiveness and usability as dependent variables.

The following dependent variables can thus be described:

- Motivation - whether the application of extrinsic motivators and competition through gamification has an effect on the ability of developers to resolve warnings.
- Effectiveness - whether the tool is effective in motivating developers to resolve warnings on a regular basis
- Usability - whether the tool is functional and easy to use for the developers.

5.1.2 Type of experiment

The type of experiment is another important factor. Ideally a controlled experiment would be performed in which there would be two groups. One group would be working with the tool, while another would be working without the tool. The results of the two groups would be compared in order to gather the results. Such an experiment would provide quantitative data based on which conclusions could be drawn. However, for such an experiment, this would be difficult. It would be impossible to measure the motivational impact without the tool's presence.

Another possibility would be to have the same experimental group work without the tool and with the tool. Such an experiment would be a longitudinal study, and only in such an experiment would the effects of the tool be effectively measurable. It would not be practical to perform either of these types of studies given the time and resources of this project.

After having considered the possibilities, a one-group pretest-posttest pre-experimental design was chosen [8]. The main difference between this type of experiment and a longitudinal study are the time factor and environment. This kind of experiment is called pre-experimental because it does not meet the scientific standards of experimental design [4]. However, it still allows us to report its findings regardless of the limited sample size [11]. Even though this type of experiment would not allow for drawing firm conclusions, it would still allow gathering valuable data for the effect of gamification on resolving automated static analysis warnings.

5.2 Pretest and Posttest

A one-group pretest-posttest is a pre-experimental design experiment in which one group of participants is tested. In a typical experiment, the effect of the independent variable on the dependent variable is examined by a dichotomy [4]. The researcher will typically compare the scenario of the experimenter being exposed to the independent variable against the one that is not exposed to the independent variable. In this type of experiment, the researcher is able to test the validity of the hypothesis by comparing the two scenarios.

In pretesting and post-testing, instead of having a control group, the same group is subjected to the independent variable. Pretesting is defined as the measurement of the dependent variables among the subjects, while post-testing is defined as the measurement of the dependent variable after the subjects have been exposed to the independent variable [4]. The subjects are exposed to a questionnaire before the independent variable is administered. The results of this questionnaire will function as a baseline. Afterward, the subjects are exposed to the independent variable. At the end of the experiment, the same questionnaire is given to the subjects. The difference between the results of these two rounds is credited to the independent variable. However, this type of experiment has the risk when participants receive the questionnaire for the second time, they may know what is being tested and respond in a way they think is desirable. These risks should be carefully considered when conducting the experiment.

5.2.1 Prestest Questionnaire

For the pretest, five themes were chosen. Each theme relates to an aspect of the experiment and the influence on the dependent variables. A quick overview of the themes is provided in this section. The results can be found in Chapter 6.

- Personal background: information about the participants such as age, education, and occupation were collected.
- Developer experience: is about collecting information about the years of programming experience and experience with Java in particular. Information about working in teams was also part of the questionnaire.
- Attitude towards software quality: participants were asked about their coding behavior pertaining to writing test, documentation, code review process, and coding conventions. The purpose of the questions within this theme was to determine the participant's attitude towards code quality.
- Attitude towards ASAT produced warnings: participants were asked if they believed ASATs contributed in upholding the quality of the code they write. Information about ASAT usage was gathered, such as how many tools are used, how often, and what step they take when warnings are encountered.
- Expectations of Checkpoint: participants were asked if gamification would contribute to resolving more warnings. Questions were asked pertaining to code review, teamwork, and the development process.

5.2.2 Posttest Questionnaire

After the pretest questionnaire, the participant takes part in the programming experiment. After having completed the experiment, the participant is presented with the posttest questionnaire. The purpose of this questionnaire is to capture the effectiveness and usability of Checkpoint. In order to determine this, a set of questions divided into eleven themes are asked.

- Experiment experience: is about acquiring information about the participant's experience of the assignment. The experience may have an influence on the results gathered for the following themes.
- UI Experience: a series of questions pertaining to the clearness of the tool. In Chapter 3 it was found that aesthetics matter when it comes to gamification. Therefore the information about the user interface and experience is gathered.
- Process Experience: whether having Checkpoint as an Ad-hoc solution was inconvenient. How many times a participant opened the tool was also asked.
- Levels: series of questions about the levels widget. Questions about the difficulty and interestingness were asked.
- Notifications: participants were asked about the frequency and effectiveness of the notifications. Notifications can be effective in capturing the user's attention. However, too many notifications may also cause a disturbance.
- Activity: questions whether the activity log and crusher chart is useful and detailed enough.
- Badges: questions about the effectiveness and competitiveness of the badges system. A question about the naming for badges was also put in to review whether the participant's found the naming scheme creative. The creativeness is part of the fun factor.
- BugsBuddy: questions about the effectiveness of BugsBuddy and its expressions.
- Leaderboard: the leaderboard shows were the current user stands among others. Questions about the usefulness and effectiveness of the leaderboard were asked.
- Checkpoint: this theme is a repeat of the expectations theme in the pretest questionnaire. The same questions of the pretest are asked again in order to determine a difference in attitude towards gamification.
- Experiment rating: a final set of questions about the quality of the experiment, the documentation, and guidance.

The posttest was followed by an interview and thus no feedback section was added to the questionnaire. The participants were provided the opportunity of giving feedback during the interview.

5.3 The Interview

In order to obtain the views and perspective of the subject, after the posttest questionnaire, an interview will be conducted. This interview will be executed in the format of a semi-structured interview. There are three types of interview: structured, unstructured, and semi-structured [24]. A structured interview allows no room for deviation. An unstructured interview, on the other hand, is also known as an informal conversational interview. Unstructured interviews are less useful when one already has a basic understanding of the phenomenon [37]. If the research goals are well-defined then it is best to use a semi-structured interview and/or a questionnaire [37]. A semi-structured interview is more suitable than an unstructured interview during this experiment because it allows us to collect data more efficiently [37].

The purpose of the semi-structured interview is to gather qualitative data. This type of interview is used when one wants to gather personal and descriptive data such as emotions, behavior, and opinions of the experiences of the participants. Such data would be difficult to gather with a questionnaire only.

In a semi-structured interview, a list of questions is prepared in order to guide the interview. Then questions are based on the goals of the experiment, in other words, they relate to the research questions. The questions should be asked in such a way that they are not biased and or not leading the participants. [1] The questions should be specific and address only one issue. Questions which have multiple hidden questions are not allowed. For example: do you use program X and Y? One should also avoid making strong opinions in the questions. Finally, the questions need to be open-ended. By asking such questions the participant is given room to give descriptive answers.

It is usual to have a few introductory questions in the beginning in order to warm up the interview and making the participant feel comfortable. There are also other points to keep in mind when conducting such an interview, such as body language and not interrupting the participant.

The interview data has to be recorded textually and may also be recorded by audio or video. During this experiment the interview was recorded textually. Since the interviews will be relatively short (fifteen minutes), taking notes is faster than recording and transcribing the audio afterwards [2]. After the interview has been conducted the data has to be analyzed. In order to analyze the results of the interview, the card sorting method will be used.

5.4 Card Sorting

Card sorting is a method to identify patterns within unstructured data [33]. Card sorting may be used for many types of projects, e.g. generating navigation for a website. For this project, card sorting will be used to identify patterns within the interview data. The concept of card sort is to have content ideas on index cards and have them sorted into groups.

[1] http://designresearchtechniques.com/casestudies/semi-structured-interviews/

[2] https://www.thoughtco.com/notebooks-vs-recorders-for-interviews-2073871

There is an open card sort and closed card sort. In an open card sort, the participants are required to first create groups of overlapping content and then label the groups. In closed card sort, the groups are predetermined and participants are required to sort the content cards. Open card sorts are more frequently used because one can learn more from them [33]. Open card sorts are not required to be completely free either. Based on what you want to learn, participants may be asked to focus on particular criteria.

Card sort consists of the following main steps: deciding what one wants to learn, selecting the open or closed method, choosing the content, selecting the participants, running the sort, and analyzing the outcome.

For this project, the open card sort will be selected since there is more to be learned from the data in this way.

5.5 The Case

In order to test the influence of the independent variable, Checkpoint, on the dependent variables, a project has to be chosen. The project for this case preferably has to be constructed in a way that is able to simulate a real-life experience and maximizes the output of the results.

Finding such a project for the experiment is not an easy task. Therefore, the project has to meet the following criteria:

- Realistic: the project should be one which one would possibly work on while on the job.
- Complexity: the project has to be complex enough such that the participants are able to get familiar and complete the tasks within the time they are provided. If the project is too easy the experiment will fail to measure the impact of the independent variable. A project which is too difficult may increase the chance of participants not completing the tasks. It is, therefore, necessary to have a sufficiently complex project such that the findings of this experiment can be generalized.
- Interesting: the project should be interesting enough such that participants feel engaged to work on the tasks.
- Tedious: in the background (Chapter 2), one of the factors of ASAT produced warnings being neglected is that the tasks are tedious. The project will therefore have to contain enough warnings such that tediousness in practice can be simulated.
- Easy to learn: since the time is limited the project should be one that participants are either familiar with or need very little time to get familiarized with.
- Technicality: the project has be in Java, because Checkstyle works only on Java source code.

After having set the criteria for the project, there was a choice to be made between creating a project which satisfies these requirements or searching for a project which already

satisfies these requirements. Creating a project complex and interesting enough to simulate a real working project would have been too difficult and unnecessary. In order to search for a project, GitHub was queried for Java projects and TU Delft projects were also looked at. JPacman [3] was finally chosen as the project the work with.

JPacman is a Java project used in a first-year course of the Computer Science bachelor program (Figure: 5.1). The purpose of this course is to teach students the value of testing. During the course, students are also required to use Checkstyle. Each commit needs to be free of Checkstyle warnings. The source code of this project is not completed and lacks key functionality of the game, such as ending the game when winning. However, in this experiment, the participants are not tasked to implement missing functionality or write tests. Their task will be to resolve the existing warnings. The source contains a custom Checkstyle configuration. In order to simulate the tediousness, the custom configuration is removed and the "sun_checks.xml" configuration of Checkstyle is used. This application of this configuration resulted in the creation of 630 warnings of 15 distinct categories according to the GDC classification [5]. The top three warnings categories were trailing spaces, final parameter warnings, java documentation issues. A full list of these issues is available in Appendix (ref).

5.6 Participants Selection

Finding suitable participants for an experiment is not an easy task. The source code of the case in this experiment is written in Java and the programming tasks will be performed within Eclipse. Due to the nature of the setup of this experiment, it is important that the subjects are at least familiar with Java and have at least two years of programming experience. Since the programming tasks will be resolving the warnings and not implementing new functionality, the participants do not need to be experts in Java. In other words, they are not required to architect and setup a Java project by themselves.

The participants were searched mainly on the TU Delft. A message was posted within the group of the Software Engineering Research Group (SERG) of Computer Science. There were also messages posted on the board within the faculty of EEMCS, and Facebook messaged were also posted within various groups. During the search for a possible job after graduation, Sogeti offered to help for the experiment participant selection. Sogeti is an information technology consulting company [4] The company offered five participants. In total eleven participants were selected, of which five from the company, four members of SERG, and two students who also had job experience. For the pilot, two students were selected.

5.7 Experiment Setup

Another important factor for the experiment is the environment in which it is conducted. To control the variables, a specific environment was set up for each experiment. As stated

[3] https://github.com/SERG-Delft/jpacman-framework

[4] https://www.sogeti.nl/over-sogeti

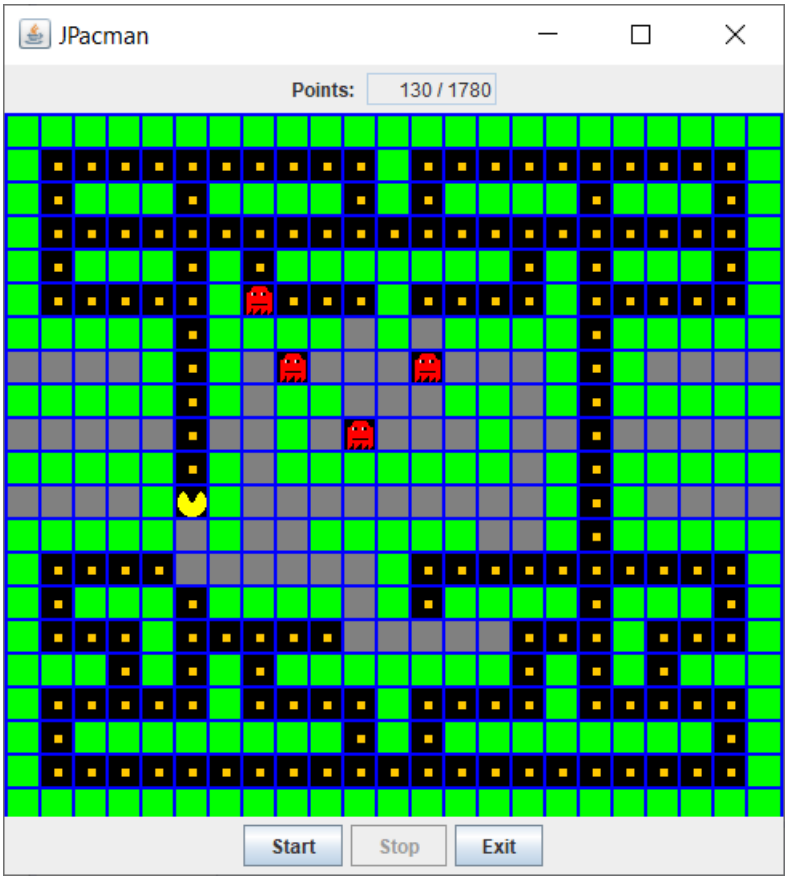

Figure 5.1: JPacman

earlier, the tool works with projects hosted on GitHub for which the GitHub application is installed and configured. For this experiment, a new GitHub organization was created. On this organization, the GitHub application for Checkpoint was installed. The application was configured for all current and future repositories in the organization. This means that applications created in the future are automatically authorized and imported into Checkpoint. After the creation of a new repository, the Checkpoint web application would receive the event and process the request. For each participant, a new repository with a clean set of code of JPacman with the configuration as described in Section 5.5 was added. By doing this, each participant had the same starting point. To save time, the creation of each repository and the initial process by Checkpoint was performed beforehand. The initial process is a first-time scan on the whole repository during which all files are saved to the database. During this process, Checkstyle is executed on every Java file. This process can take some time depending on the size of the project. The same protocol was followed during the pilot run.

The Eclipse IDE version oxygen March 2018 was installed in which the participants had to perform their tasks. The project was also already added within the workspace. Partici-

pants were only required to open the project. Checkstyle version 8.11 was already installed within the IDE.

A GitHub desktop application was installed and opened. Participants were required to use the application for committing and pushing changes. The Checkpoint web application was available in Firefox Developer edition.

Checkpoint was running on the machine of the observer using a XAMPP server with PHP version 7.2.3, Apache 2.4, MariaDB server version 10.1.31, and Checkstyle 8.11. To redirect the webhook events of GitHub to the localhost server on the machine, Smee was running in the background. The IDE, GitHub desktop application, and Firefox were all running in a separate desktop view on Windows 10.

The experiments were conducted in quiet spaces. Only the participant and experimenter were present during each run. During the experiment, each participant's actions were observed. Internet searches during the experiment were allowed for task-related tips and tricks. The participants were also nudged by the observer to make sure they followed the tasks. These nudges were verbal, reminding the participant to keep moving forward. An example of such a nudge is to remind the participant to commit their change when they had already resolved a significant amount of warnings.

5.8 Pilot

To minimize the chance of errors in the experiment, a pilot was run. For the pilot, two people were selected. One was a non-programmer student who studied Computer Science for one year, while the other was a Computer Science student. A pilot is a good way to fine-tune an experiment design. The students provided valuable information regarding the user experience and user interface from different perspectives. During the pilot runs each gamification widget was assessed. Furthermore, some bugs were discovered during the pilot and fixed afterward.

- The initial idea was to have each user log in using their GitHub account. During the experiment, it became clear that this was not such a trivial task. Changing a GitHub account on the machine and switching it back requires a bit more work than logging in and out of the GitHub application. One has to clear the old account via the command line and add the new one. This process had to be repeated before every run and was time-consuming. Since there was no need for the experimenters to use their own account, this part of the process was changed. Thus simplifying the start process of each run.

- Another issue discovered pertained to the empty graph view of the warnings resolved. After registration, the graph had no commit points and showed nothing. After the first commit, the first y-coordinate (amount of warnings resolved) would show up right on the 0-axis. This behavior confused users about the purpose of the graph. This issue was taken care of by adding empty commits to the graph when the number of commits was less than the span of the x-axis. Secondly, an explanation to this widget was added explaining its purpose.

- The activity log was showing achievements, but not in detail. The testers wanted to see more details in the activity log. Why a user was being promoted and which warnings were resolved in each activity was missing from the timeline. These points were fixed by adding a more detailed description of each activity log item. The warnings resolved activity showed a small summary of warnings fixed. The badge itself was also put in the activity log.

- Competition was a key element in the gamified platform and being able to see the progress of others would have been valuable. Thus the badge system was expanded by showing how many other players unlocked the badge.

After the pilot, the user interface of the program was updated based on the feedback of the pilot. The process of the experiment was changed by removing unproductive steps such as adding the user to a repository and logging them in on the machine. However, not all feedback was dealt with. A point was made about showing the progress of friends in the graph. The idea was to show the progress of friends alongside the progress of the user to increase motivation. Considering the nature of the experiment, the only possible comparison would have been among individual commits between the users. Comparing individual commits in the graph of warnings resolved is not a fair comparison. Each commit is different, has a different purpose, is committed at a different time and can therefore not be compared in a meaningful way. The progress could have been compared over a longer span, such as a week. This could be achieved by showing the total or the average number of warnings resolved per week and plotting the progress of friends in the graph. Due to the nature of the experiment, it was not valuable to show the progress of friends within the graph.

5.9 Execution of the experiment

The experiments were conducted on seven days in January 2019. Each experiment was conducted in a room with only the participant and the observer. A small procedure of the experiment was developed and can be found in Appendix (B). Before each run, a new repository with the JPacman code was created. All repositories were processed with Checkpoint beforehand to save time. Each participant was assigned an experiment code. The same code was used for the repository naming.

At the start of each experiment, the participant was welcomed and an informal introduction took place. After the introduction, participants were presented with a document providing the necessary explanation for the experiment. Participants were required to complete the pretest questionnaire. Twenty minutes were given to each participant for the programming part of this experiment. The documentation of the tasks can be found in Appendix (A). Due to the short amount of time, participants were guided through the start process of the programming tasks. Such as familiarizing participants with the GitHub desktop application and Eclipse IDE. Each participant was asked if they were comfortable with being observed during the programming tasks. None objected to being observed. Before the end of the twenty minutes, participants were reminded to make one last commit.

During one experiment, the last commit of the participant was not processed due to internet connection failure. This issue was resolved afterward. In this case, the participant did not mind this problem. All other experiments went well without any notable issues.

After completion participants were required to fill in the posttest questionnaire. The experiment was concluded with a fifteen-minute interview. The total length of the experiment was one hour. All participants finished their tasks within the time limit.

Chapter 6

Results

In this chapter the data of the pretest and posttest are discussed. The discussion about these results will be performed in Chapter (7).

6.1 Participant Profile (Pretest)

To get an understanding of each participant several questions were asked about themselves, their experience and attitude towards software quality.

Personal Background The participants were asked about their personal background first. The participants were recruited at the Technical University of Delft, and at Sogeti. All the participants were above the age of twenty. Eight participants were below thirty, while three were above thirty years of age (Fig. 6.1). All the participants had at least a Bachelor level degree or in pursuit of one (Fig. 6.2). Only one participant did not have a Bachelor degree, but was still studying and already had work experience. Since half of the participants were from the SERG, it is not a surprise that there were two post-doc researchers, one Ph.D. student, and an assistant professor (Fig. 6.3). All the participants from the company were working as back-end developers.

Developer Experience Seven out of eleven participants had between two and five years of development experience. Four participants had more than five years of development experience (Fig. 6.4). The participants found themselves averagely experienced *(3.0)* with Java (Fig. 6.6). Only one participant said not to be experienced with Java while still having three years of development experience. The participants of TU Delft, on average, found themselves more experienced in Java than participants from Sogeti. This result is not such a surprise. According to the participants, the use of Java is quite common at the TU Delft. The results pertaining to years of development experience and average experience with Java meets the requirements for the participants shown in Chapter 5. The participants' experience of working in teams resulted in an average score of *3.5* (Fig. 6.7). Only two developers were not familiar with Eclipse as an IDE (Fig. 6.5). Nine developers were familiar with Eclipse. Taking into account the years of development experience the developers had, this was not

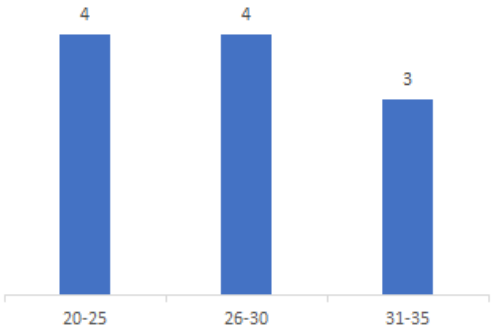

Figure 6.1: Age Distribution

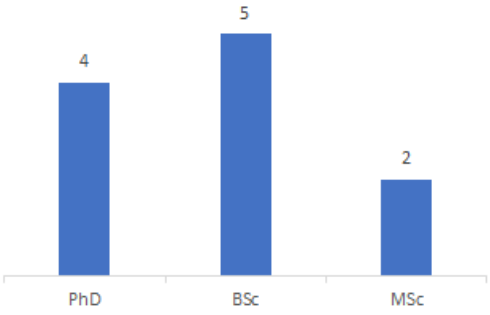

Figure 6.2: Education Levels

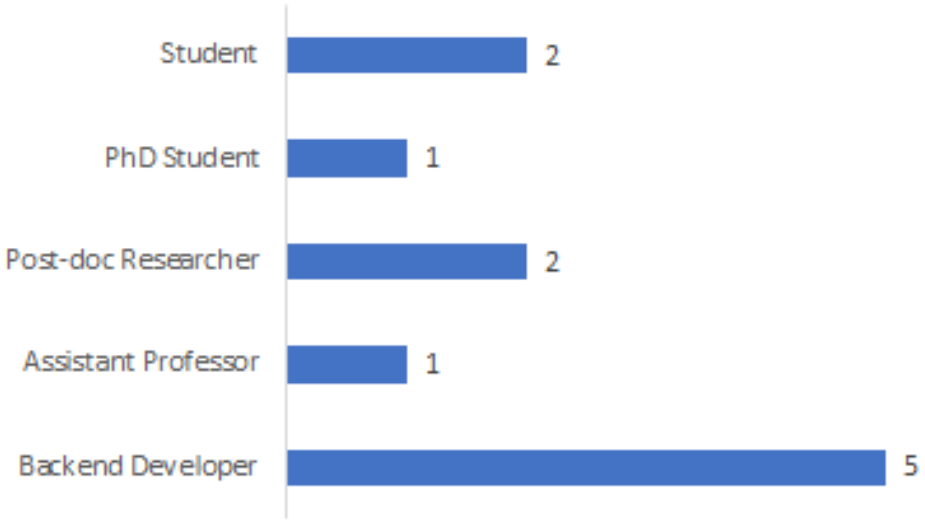

Figure 6.3: Occupation

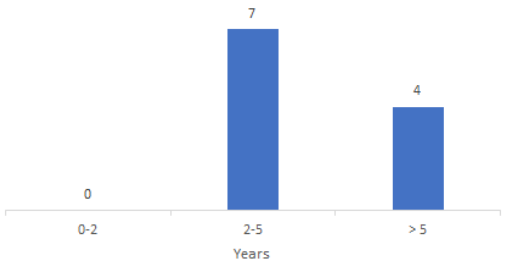

Figure 6.4: Programming Experience in Years

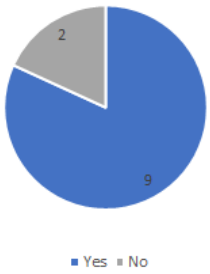

Figure 6.5: Familiarity with Eclipse

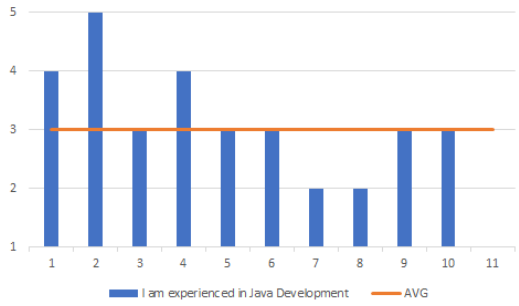

Figure 6.6: Experience with Java

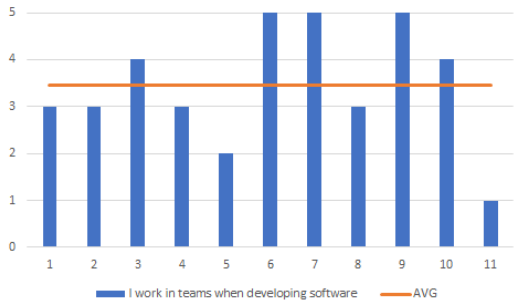

Figure 6.7: Experience working in Teams

an issue. The two participants who did not user Eclipse only required extra guidance in finding shortcuts within the IDE.

Attitude towards Software Quality The following questions were asked about the participants' attitude towards code quality and resulted in average scores of:

- Functional code is more important than clean code *(*2.6)
- When programming, I write documentation for all code *(*2.8)
- When programming, I write tests for all code *(*2.8)
- I find the code review process useful *(*4.2)
- I obey to the coding conventions when programming *(*4.4)

The results are also available in Figure 6.8. On average the participants did find the code review process to be useful, and they also obey to coding conventions. When asked whether they think that functional code is more important than clean code, 45% of the participants responded with neutral and 36% responded with less than the neutral position. This shows that on average, the participants do not think that functional code is more important than

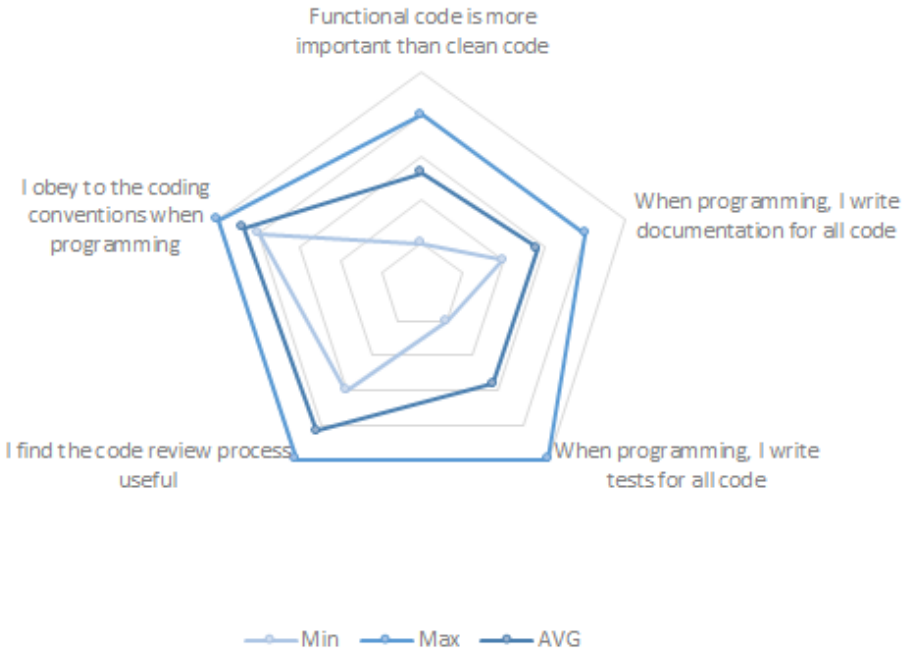

Figure 6.8: Attitude towards Code Quality

clean code. The results also show that participants do not write tests or documentation for the code at all times.

Seven out of eleven participants answered *yes* when asked if there is a code review process in place in the projects they are working on. Six out of eleven participants said that there are *no* coding conventions in place for the projects they work on.

Automated static analysis tools In this section the following questions were asked about the participants' attitude towards code quality.

- Automated Static Analysis tools (ASATs) contribute in upholding the quality of the code I write (3.5)
- Warnings generated by ASATs are completely useless (2.1)
- How many Static Analysis tools do you use at one given time?
- I use Automated Static Analysis tools to inspect my code (during)
- When an ASAT generates warnings I tend to (do the following)

When asked whether ASATs contribute to upholding the quality of writing source code, the average score was 3.5. This result shows that on average the participants do agree that

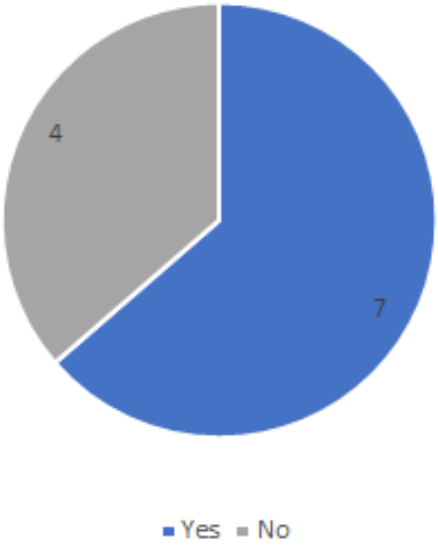

Figure 6.9: There is a code review process in the development process of the projects I work on

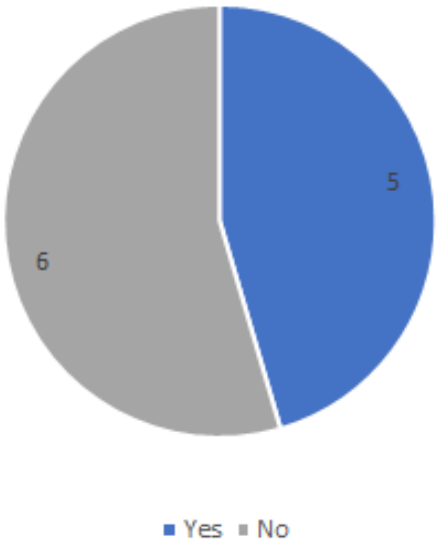

Figure 6.10: There are coding conventions in place for the projects I work on

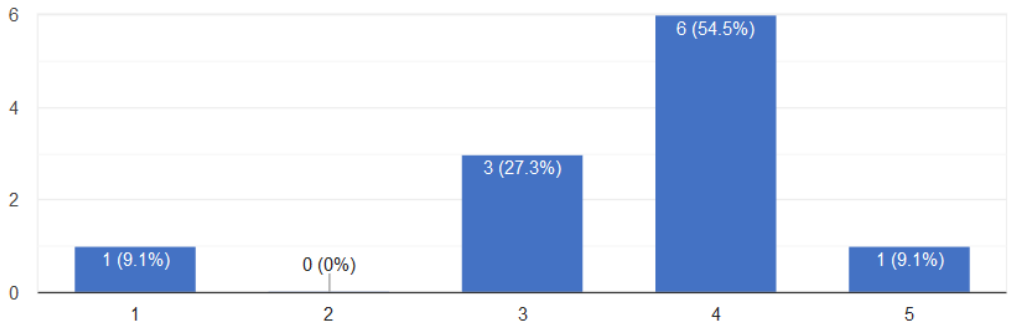

Figure 6.11: Automated Static Analysis tools (ASATs) contribute in upholding the quality of the code I write

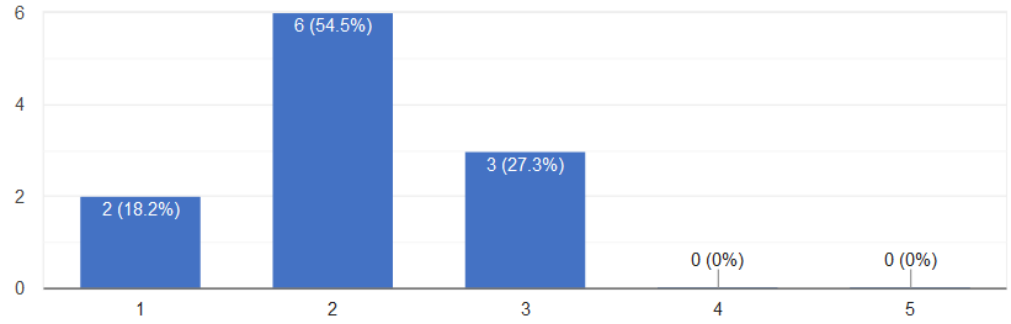

Figure 6.12: Warnings generated by ASATs are completely useless

ASATs play a role in the quality of the source code (Fig. 6.11). One participant totally disagreed with the question.

None of the participants agreed when asked whether warnings generated by ASATs are completely useless (Fig. 6.12). The scores resulted in an average score of 2.1.

45.5% of the participants disclosed to not use automated static analysis tools. 54.6% of participants stated to use one or more ASATs at a given time (Fig. 6.13).

In Figure 6.14 the results show that 27.3% of the participants never use ASATs. 72.8% of developers disclosed using ASATs during development. Half of the developers using ASATs disclosed only using ASATs just before committing the source code.

Figure 6.15 show what the participants tend to do when an automated static analysis tool shows warnings. The majority of participants, 50% tend to resolve warnings pertaining to their code only. Another 33% tends to only resolve warnings they consider important. One participant disclosed to resolve all warnings provided by ASATs.

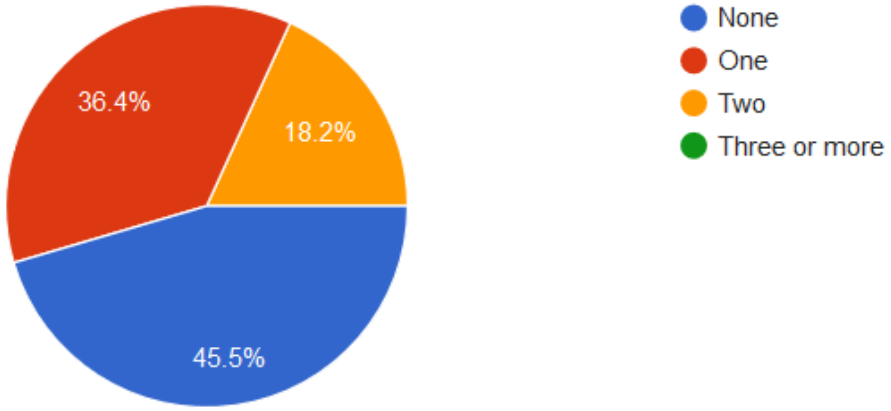

Figure 6.13: How many Static Analysis tools do you use at one given time?

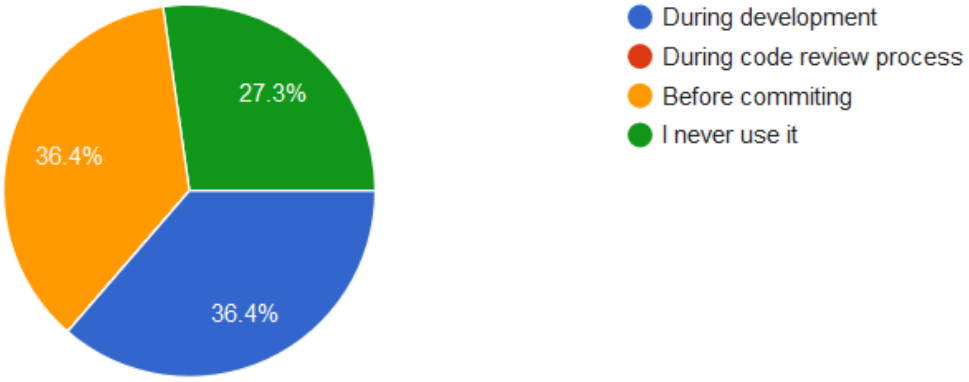

Figure 6.14: I use Automated Static Analysis tools to inspect my code during?

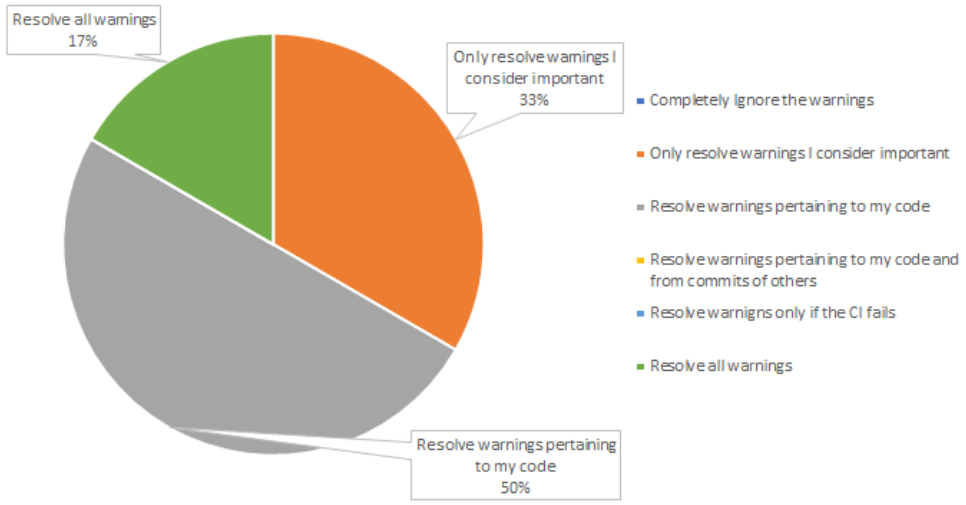

Figure 6.15: When an ASAT generates warnings I tend to?

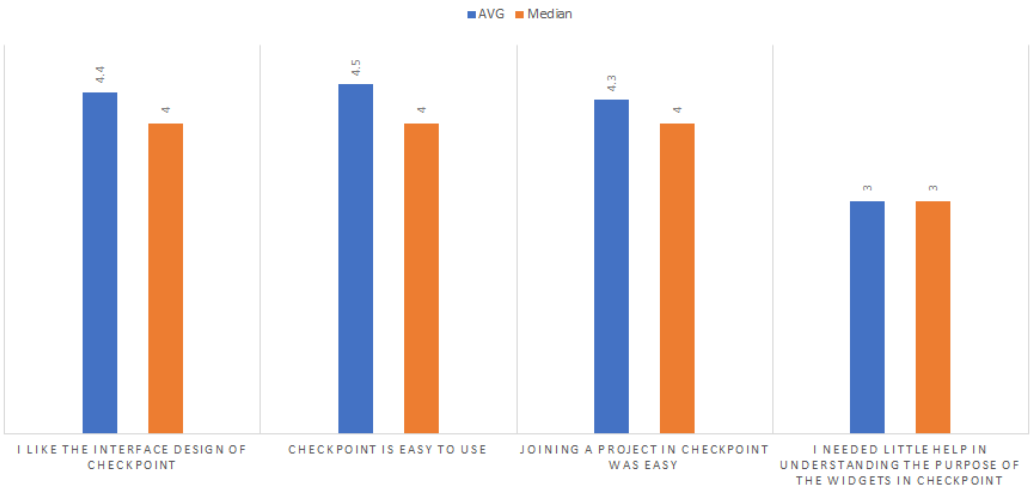

Figure 6.16: UI Experience

6.2 Checkpoint (Posttest)

In this section, the results of the posttest will be presented. For each mechanism in Checkpoint, there is a section in the posttest addressing the usefulness and creativity factor of the widget.

Figure 6.16 show the results for the user interface and user experience. Figure 6.17 and 6.18 show the results for Checkpoint with respect to the developer experience.

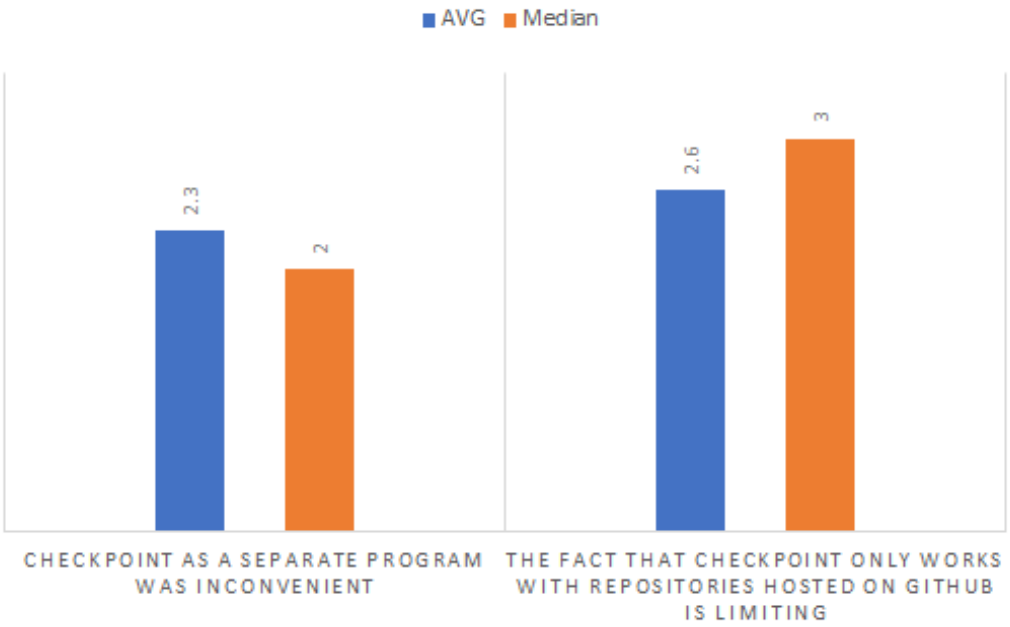

Figure 6.17: Process Experience

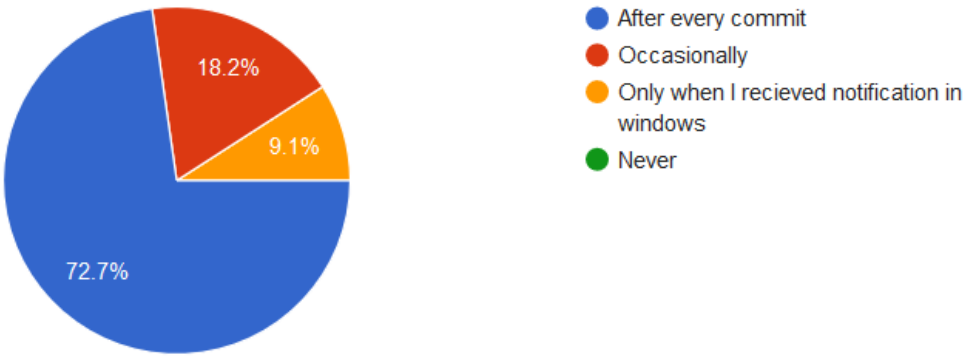

Figure 6.18: Triggered to open Checkpoint

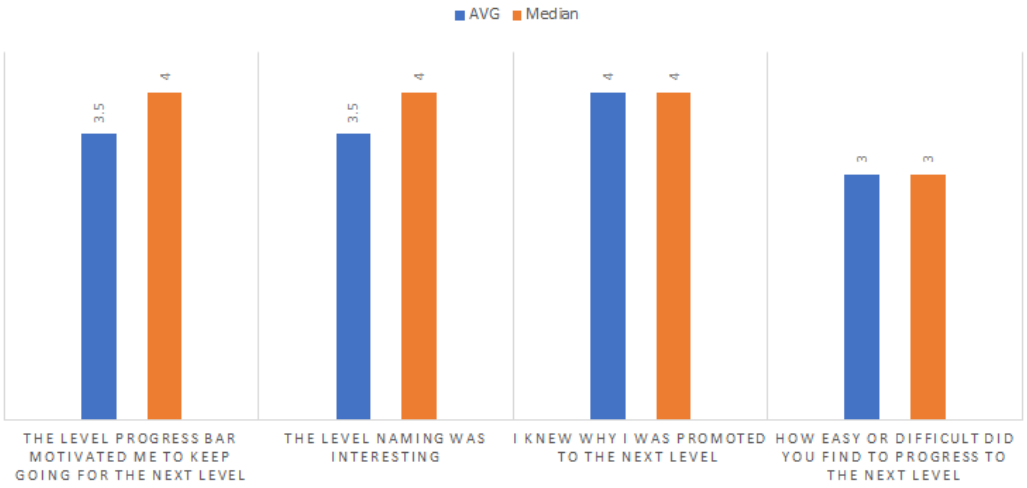

Figure 6.19: Levels

Levels The following questions were asked about the levels mechanism:

- The level progress bar motivated me to keep going for the next level *(Motivation)*
- The level naming was interesting *(Fun)*
- I knew why I was promoted to the next level *(Feedback)*
- How easy or difficult did you find to progress to the next level *(Challenge)*

Notifications The following questions where asked with regard to the notifications:

- The push notifications from Checkpoint were a disturbance to my workflow *(Process)*
- The push notifications from Checkpoint awakened my curiosity *(Motivation)*
- The feedback from Checkpoint through notifications was frequent enough for me *(Feedback)*

Activities Log The following questions were asked about the activities log mechanism:

- The Activity log is a useful widget, it provides me information about my progress *(Feedback)*
- The activity log was detailed enough *(Feedback)*
- The Crusher Statistics (warnings resolved chart) is a useful widget as it provided me information about my progress *(Feedback)*

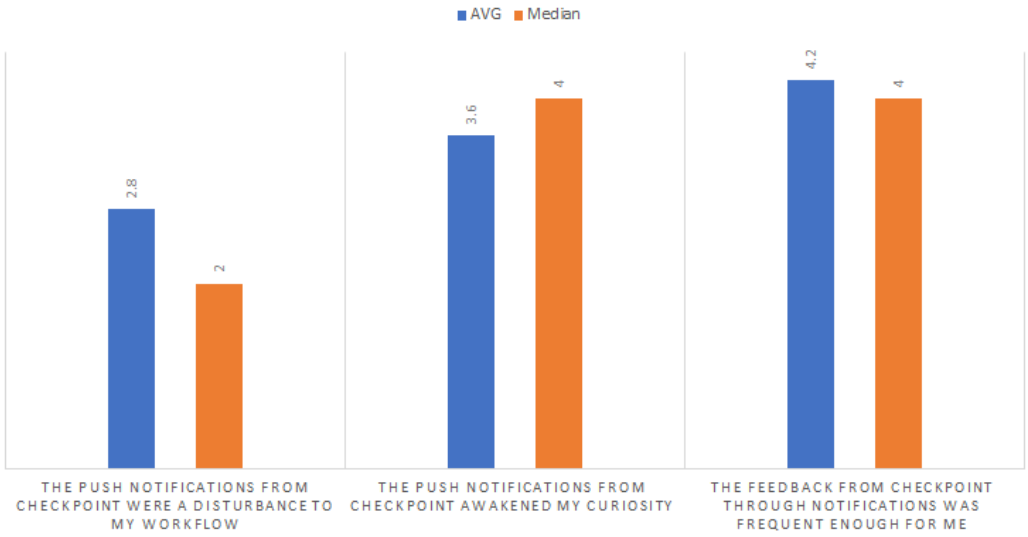

Figure 6.20: Notifications

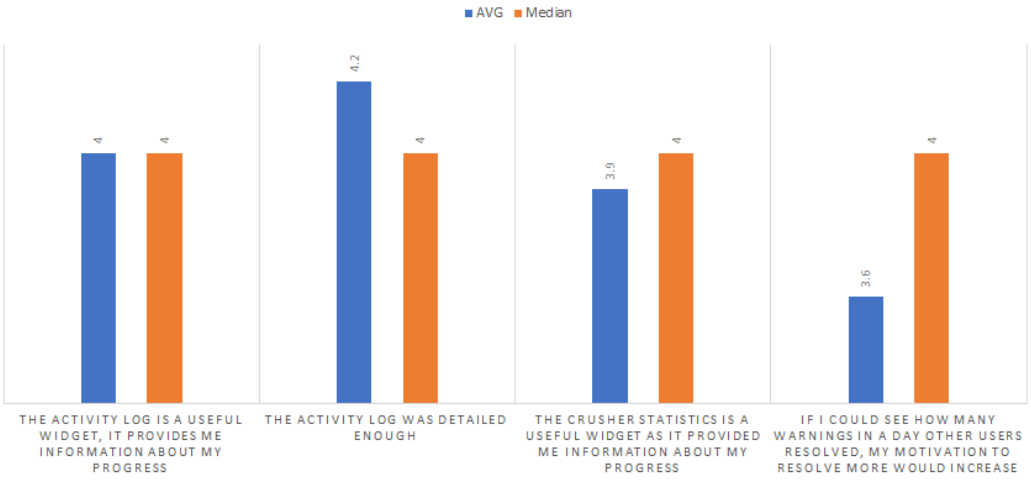

Figure 6.21: Activities Log

- If I could see how many warnings in a day other users resolved, my motivation to resolve more would increase *(Competition)*

Badges The following questions were asked to measure the usefulness of badges:

- Badge Achievements motivated me to resolve more warnings in order to unlock them *(Motivation)*

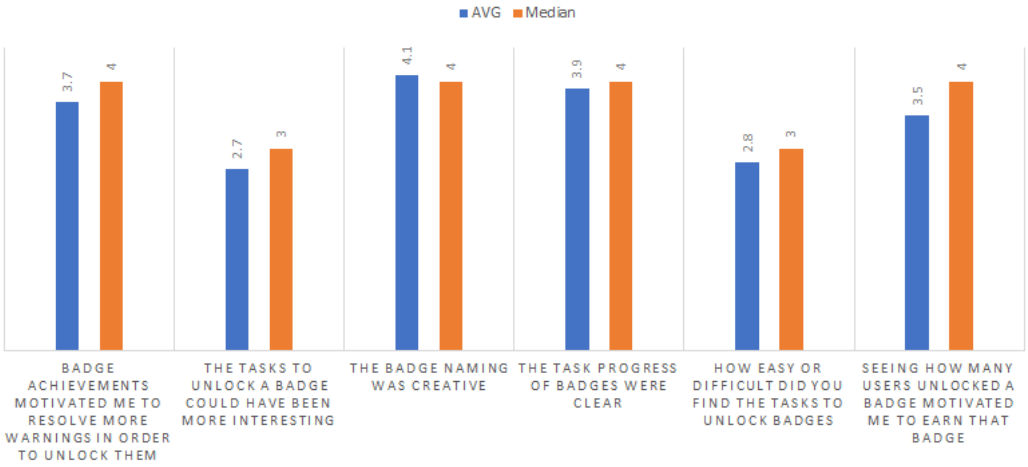

Figure 6.22: Badges

- The tasks to unlock a badge could have been more interesting *(Fun)*
- The badge naming was creative *(Fun)*
- The task progress of badges were clear *(Feedback)*
- How easy or difficult did you find the tasks to unlock badges *(Challenge)*
- Seeing how many users unlocked a badge motivated me to earn that badge *(Competition)*

Leaderboard The following question were asked about the leaderboard:

- I found it useful to see were other players rank in the game *(Competition)*
- The leaderboard motivated me to gain XP *(Motivation)*
- If I had a leaderboard with my friends, it would motivate me more than having just a global leaderboard *(Competition)*

BugsBuddy During the experiment, none of the participants had noticed BugsBuddy. No participant's action resulted in BugsBuddy being sad. Due to these reasons, the questions for BugsBuddy were scrapped from the results except for one question. The question "I found the idea of having a virtual buddy creative" will still be addressed in the results and discussion. This question addresses the need for and creativeness of the mechanism. The score for this question resulted in an average of 3.5 and a mean score of 4.0. Two participants disagreed with having BugsBuddy.

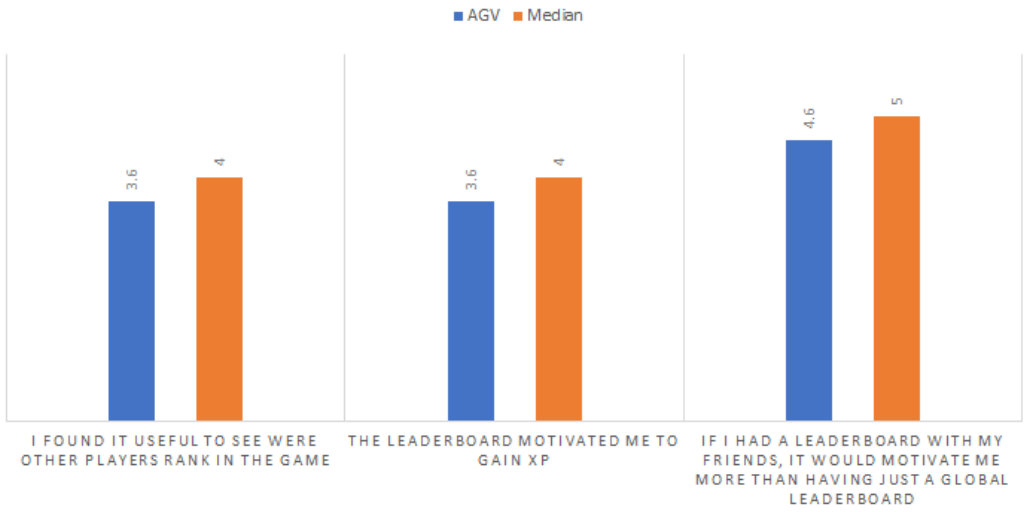

Figure 6.23: Leaderboard

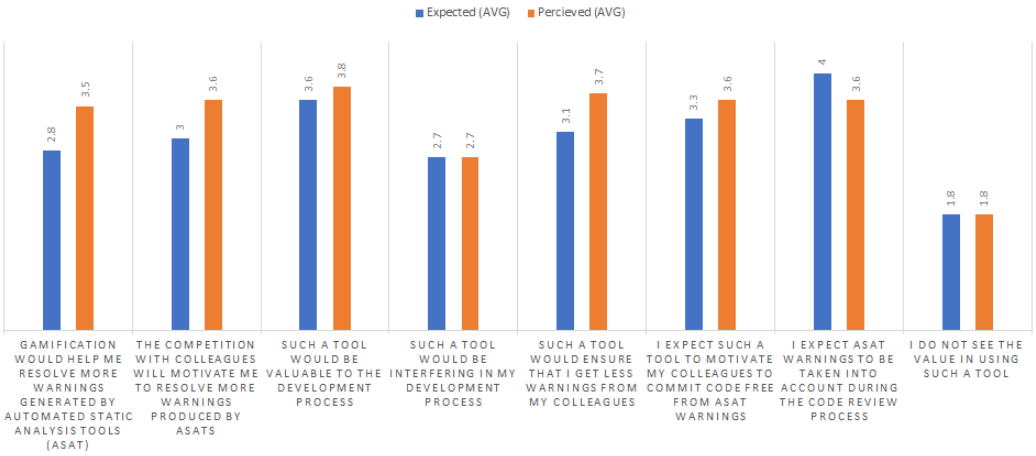

Figure 6.24: Expectations (average)

6.3 Expectations on Gamification

In figures 6.24 and 6.25 the results for the expectations of gamification and the results after having worked with Checkpoint can be found. The results show no drastic changes between the expectation and perceived value. The results are, on average, in favor of gamification. The results show that after being exposed to Checkpoint, the participants changed their opinion about gamification having an effect on their motivation (mean of 3 becoming 4 in the posttest).

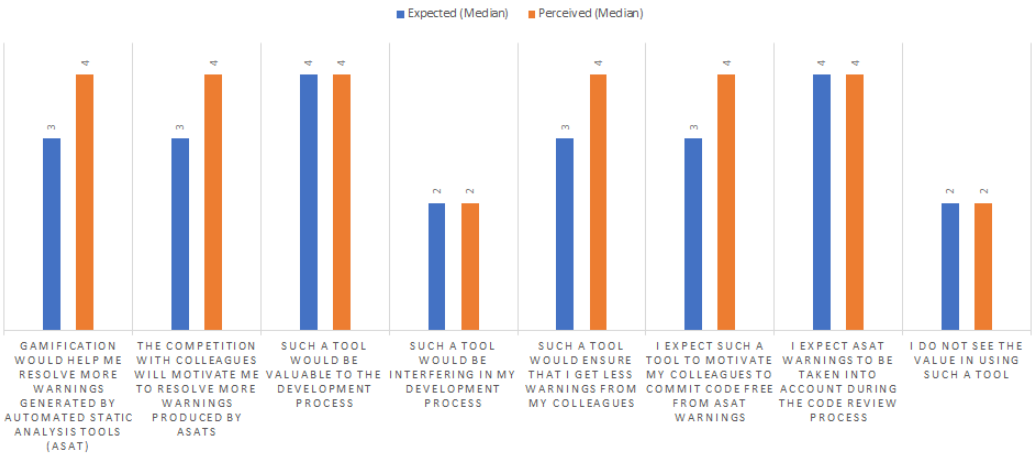

Figure 6.25: Expectations (median)

6.4 Experiment Experience and Rating

In the first section of the posttest questionnaire, the participants were provided the opportunity to provide feedback about the experiment's experience (Fig. 6.26). The experience addresses: difficulty, time pressure, interest, and guidance. The results show that the time pressure and difficulty of the assignment were low. The participants found the assignment to be interesting resulting in an average score of 4.2. The guidance for the assignment was more than enough resulting in a score of 4.7.

In figure 6.27 the scores for the overall rating for the experiment can be found. The results show a high rating for all questions. The quality of the project was also rated high with an average score of 4.2.

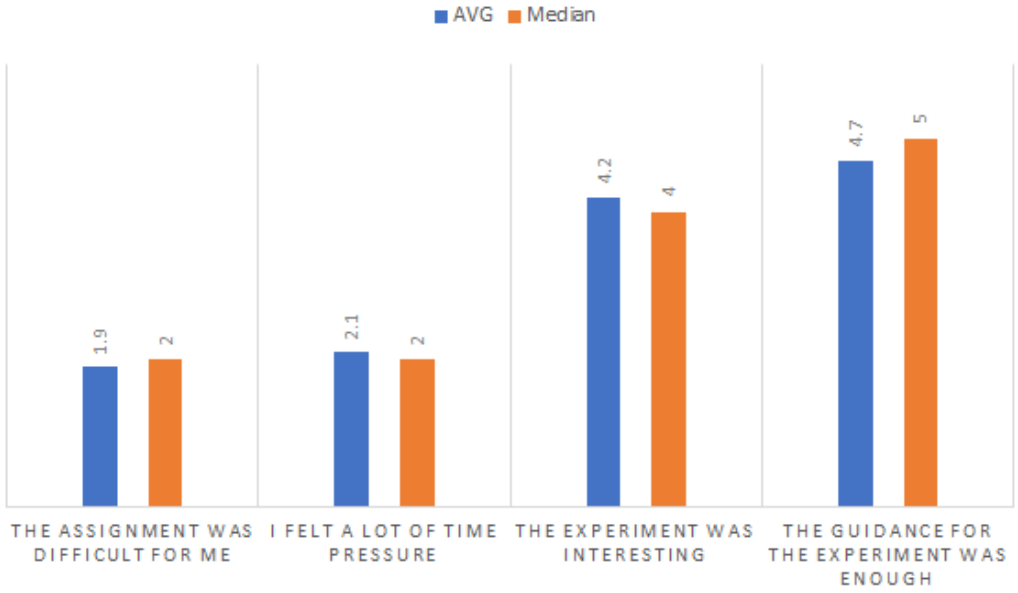

Figure 6.26: Experiment Experience

Figure 6.27: Experiment Rating

Chapter 7

Discussion

The purpose of the questions asked was to test the dependent variables in the experiment. The variables are motivation, effectiveness and usabiltiy as stated in Chapter 5.

7.1 Motivation

Motivation is a key element in a gamified environment. To investigate whether the tool is adequate to motivate developers the following questions were asked:

- 12. The level progress bar motivated me to keep going for the next level (3.5)
- 17. The push notifications from Checkpoint awakened my curiosity (3.6)
- 23. Badge Achievements motivated me to resolve more warnings in order to unlock them (3.7)
- 28. Seeing how many users unlocked a badge motivated me to earn that badge (3.5)
- 34. The leaderboard motivated me to gain XP (3.6)
- 36. Checkpoint motivates me to resolve more warnings than I usually do (3.5)
- 37. The competition with colleagues will motivate me to resolve more warnings produced by ASATs (3.6)
- Interview Q4. Do you think Checkpoint would motivate developers in resolving ASAT warnings

The purpose of each game mechanism in Checkpoint is to motivate the developer. For each mechanism, a question relating to motivation was asked.

With an average score of 3.5 as presented in Figure 6.19, the results show that the level widget is interesting enough to nudge developers. One participant scored the question with a one, expressing that the level bar does not play a role in motivation at all. The purpose of the notifications was to nudge developers into opening the dashboard and reviewing their

latest achievements. The score for notification's curiosity suggests that the notifications do grab the developers attention. The scores for badges and leaderboard also suggest playing a positive motivational role.

When asked whether Checkpoint would motivate developers in resolving ASAT warnings, the participants gave an average score of 3.5. The score in the pretest averaged 2.6. This result shows that the participants' opinion did change in favor of gamification after being exposed to Checkpoint. One participant did give a score of one. This participant stated in the interview that there should be a balance between gamification and seriousness. The participant stated that at the moment the application is too playful being centered too much around gamification. The seriousness of the application could be increased by showing metrics with respect to business goals.

As far as competition is concerned, the average score is 3.6. Many participants find common-ground that competition will motivate them to resolve more warnings. One participant even stated that "It is Human nature to compete, whether we like to admit it or not". However, many participants also stated in the interview that developers should be careful of competitions. The competition among developers in a team may cause developers to lose focus. Some participants feared that developers may resolve warnings only for the sake of gaining points.

7.2 Effectiveness

The effectiveness of the tool lies in its ability to stimulate its users to achieve the intended goals. As stated before, the purpose of Checkpoint is to motivate developers into resolving ASAT produced warnings. In a short experiment, it is difficult to measure the effectiveness of the program. To still get an impression of the effectiveness of Checkpoint, questions, whether Checkpoint would support warning reduction by colleagues, were asked.

- 40. Checkpoint would ensure that I get fewer warnings from my colleagues (3.7)
- 41. I expect Checkpoint to motivate my colleagues to commit code free from ASAT warnings (3.6)
- 42. I expect ASAT warnings to be taken into account during the code review process when Checkpoint is used (3.6)
- 38. Checkpoint would be a valuable tool for my development process (3.8)

When asked if Checkpoint would ensure that developers get fewer warnings from colleagues the score resulted in 3.7. In the pretest, the score resulted in 3.1. The result from the questionnaire shows that the expectations of the participants changed in favor of Checkpoint. The second question asked whether colleagues would be motivated to commit code free from warnings, the score rose from 3.3. to 3.6. These results can be found in Chapter 6. The results of these questions show that the participants expect their peers to be motivated to resolve warnings due to Checkpoint.

The impressions of the participants using Checkpoint during the code review process did tumble from 4 to 3.6 (Fig. 6.24). The result shows that the participants' expectation of Checkpoint playing a part during the code review process was not as the participants expected.

Overall the impression of Checkpoint was quite good. When asked if Checkpoint would be valuable for the development process, the scores rose from 3.6 to 3.8. During an interview, a participant stated to find Checkpoint useful, especially for projects which have been around for a while. During the interview, participants were asked if they see themselves using Checkpoint. Eight out of eleven said that they would use the tool. One participant stated only to use the tool if team members used it. The other three participants did not agree to use the tool on their own project. These three participants were not keen on using gamification and competition to support business goals. Two of those participants stated not to like competition. One participant stated to not care about competition and not being a fan of gamification. The concern here was that developers may be looked down upon when they have resolved fewer warnings. Even participants who said to like the idea of gamification warned against having competition. One participant worried about the results being used as a metric of developer performance.

The majority of participants liked the idea of having gamification and competition. One participant stated that Checkpoint would only be useful in the long run if there was an incentive from the company they work in. Having period rankings in the leaderboard would also be effective. For example, having a weekly ranking.

The results show that one should be aware of unfavorable competition and possible abuse of the system. Overall, participants do think that Checkpoint can be effective in reducing ASAT produces warnings.

7.3 Usability

The last variable tested, usability, addresses the ease of use and functionality of the tool. The participants were asked questions about the User Interface, and the process experience.

In Chapter 3 it was shown that the whole experience of the tool including the user interface play an important part in gamification. Therefore, the first question was to ask the participants whether they liked the interface of Checkpoint. The average score was 4.2. The result show that the participants liked the interface. Using Checkpoint was also easy (4.5). Joining a project was also well experienced, resulting in a score of 4.3.

When asked whether the participants needed help in understanding the widgets, the score was an average of 3.0. Five participants even gave a score 2.0. These scores show that participants needed more help than expected in understanding the purpose of the widgets.

Having Checkpoint as separate program was not much of an issue. Ad-hoc gamified solutions may have a lesser impact as discussed in Chapter 2. Therefore, it was of significant value to test whether Checkpoint would be less effective as a separate program. Two participants gave a score of 2.0 and another two a score of 4.0. Although the result show that an ad-hoc tool is not completely inconvenient, the tool in not completely exonerated.

These numbers show that four participants did experience some inconvenience in having the tool as ad-hoc. A further investigation would be needed to resolve such an issue in practise.

Eight participants stated to open the tool after every commit. This result shows that participants were interested and anticipated the feedback they would receive from Checkpoint after they committed code.

When asked if Checkpoint was limited by the fact it only work with GitHub, most participants disagreed (2.6). This result shows that participants do not mind that the tool work with repositories hosted in GitHub only.

7.4 Mechanism

To measure the effectiveness of each widget, questions concerning each has been asked in the questionnaire.

Notifications The Notifications are part of the feedback loop that keeps nudging the user. As long as the application is running, whenever points are gained or achievements are unlocked, notifications are pushed. The notifications are part of a loop that is supposed to motivate the user by providing a nudge and feedback. Notifications also have an impact on the usability of an application. Users may perceive notifications as annoying if they are pushed too frequenty.

When asked whether the push notifications were a disturbance, the average score was a 2.8. Four of the eleven participants gave a score of 4.0. The results show that the frequency of the notifications should have been a little lower. A few participant's attention was grabbed when after their first commit, they had three or more notifications at once. Possible solutions here could have been to group notifications.

One a positive note, the notifications did awake curiosity (3.6) and the feedback from the notifications was enough (4.2). These two results show that although the frequency was on the edge, the notifications did fulfill their tasks of nudging the user.

Levels The levels widget is used to provide the user with the sense of a reputation. As XP points are gained, a user is promoted to a higher level. The name of the level is also shown in the leaderboard. The Levels in this application also embody a fun factor by using a creative naming scheme.

The levels were appreciated in general. When asked if levels motivated the participant to progress to the next level, the score was averaged 3.5. The same score is also given to the creativeness of the level naming. The results show that the levels were motivational, creative and the users knew why they were getting promoted to the next level. Only one participant indicated not to be motivated by the levels.

Badges Badges were another interesting widget of the application. The badges were given creative names in an attempt to incorporate a fun factor. According to the results, the participants agreed that the naming for badges was indeed creative (4.1). During the experiment, some participants even laughed after reading some of the badge names. The badges also

motivated the participants to complete certain tasks and gain points (3.7). By showing how many other developers had achieved a badge, an element of competition was added. Results showed that this had an impact on the motivation of participants. The score for this question was an average of 3.5. The participants also found the progress of the tasks to be clear enough.

The tasks were also interesting enough. When asked if the task naming could have been more interesting, the users disagreed with an average score of 2.7. The participants found it to be relatively easy to unlock badges. For this short experiment, this result is not an issue. However, if this application were to be put in production and used for a longer time, this may cause an issue. Easy badges may become boring. The effects of this could only be tested in a long-running experiment.

Leaderboard The leaderboard was an interesting widget in the sense that it is the most competitive one. The leaderboard shows of where a participant stands among others. The participants agreed to find it useful to see where other players rank and also stated to be motivated by it. Due to the nature of this experiment a global leaderboard was shown. The participants were asked if they would have been more motivated if instead they were shown a leaderboard with friends. The participants completely agreed that this would be more effective. Even though many participants noted to like the leaderboard and the competition, they were also concerned. One participant stated that instead of resolving relevant warnings, one might start resolving warnings for the sake of points or beating a teammate on the board.

Activity Log The activities of the user are shown in two sections. The activity log on the right of the screen shows the activities of the user. A chart on top of the screen shows how many warnings per commit are resolved. These widgets' main purpose is to provide feedback. The activity log was best received out of all widgets. This can be concluded based on the results of the scores. The usefulness of the widget scored quite high (4.0). The participants also rated the activity log to be detailed enough. One participant did not like the activity log. The participant found the log to be too centered around gamification instead of the commit itself. Even though this was the purpose of the tool, there is also a point here. The gamification tool needs to balance playfulness and seriousness.

The chart showing how many warnings were resolved was also well appreciated. The participants rated its usefulness with an average of 3.9. Participants also stated that the chart could be more motivational if the warnings resolved of other users were also shown in the chart. As stated earlier, this is something which could not have been implemented in this experiment. The results do show that users wish to see more competition.

7.5 Ignoring ASAT Warnigns

RQ1: Why do developers ignore warnings produced by ASATs? In Chapter 2 the background literature is presented. The background literature explored the current state of ASAT produced warnings.

The literature study found the following:

- Overload of warnings.
- Relevancy of warnings.
- Some less important types of warnings are ignored.
- High amount of false positives.
- Lack of explanation.
- Lack of quick fixes.
- Old bugs not being fixed.

These are some of the reasons which studies have found contributing to the lack of developers resolving warnings. The overload and importance of warnings lead to developers ignoring all warnings altogether. The longer a bug stays in the system, the more trusted the bug becomes. Trusted code is rarely updated because it is believed to be stable. This results in old warnings which forever stay in the code base. The lack of explanation and quick fixes for some warnings may also be the cause of the developer ignoring the warning. This is not at all surprising if the developer does not understand the warning, it becomes difficult to fix it.

7.6 Existing Gamification Models

RQ2: What existing gamification models for improving software engineering processes exist? In Chapter 3 various gamification mechanisms and techniques which have been investigated are presented. The subject explored in the Chapter is significant, but its application within software engineering is still a challenge. Designing and implementing an application for gamification during code development is a serious task. One must make sure that the seriousness and playfulness are balanced. Therefore, it is significant to have a guideline for applying gamification within SE.

During the research, the GOAL methodology was identified. The GOAL(Gamification focused On Application Lifecycle Management) methodology works by identifying the objective of the gamification, performing a player analysis, and defining a gamification scope. Based on these criteria the game is designed. This methodology was proven to work and the only extensively detailed methodology on this topic.

7.7 Gamification within Software Development

In this section, RQ3 is explored. **RQ3**: Are developers willing to use gamification within their software development process in order to resolve ASAT produced warnings? This question addresses the issue of developer behavior. Are developers ready to alter their daily development routine to include Checkpoint? Do developers see themselves using a tool like Checkpoint? What are the thoughts of developers using gamification in general within software development?

During the interview developers were asked the following: "*Do you see yourself using a tool like Checkpoint when working on your projects?*". Three participants stated not to use Checkpoint on the projects they are working on. The remaining eight participants said "yes" to using Checkpoint. Two of those participants had an additional statement to their "yes". One participant said that the use of the tool depended on company policy. The policy was needed in order to ensure that team members would actually use the tool. Another participant stated only to use the tool if others are participating. This result is not a surprise, the tool is built around various gamification elements including competition. The gamification effect is stronger if a connection between players is formed. One could state that working in the same team is a significant connection.

Three participants also stated "I can see my friends using this". This statement shows how powerful the meaning of connection is within gamification.

The following question was also asked: "*7. What are your thoughts about gamification within software development?* " This questions had a low priority during the interview and was asked to five participants. The purpose of this question was to gain insight from the participants about gamification in general. Of those five participants, two answered "yes" to the previous question. These two participants stated that this is a good and fun idea. The three participants who had answered "no" to the previous question, stated to have concerns. One common statement was that they saw risk, even though they recognized something positive in gamification. The risk that developers might get caught up in the game of gaining points. These participants evaluated that the risks are greater than the benefits. This experiment is not suited to conclude whether or not the risk outweighs the benefits.

The answers to these questions show that developers do think that gamification within software development is fun and they are willing to use it. They do, however, see some risks. These risks should be taken into account and monitored when the tool is applied in practice.

7.8 Extrinsic Motivators

In this section **RQ4.1**: Do extrinsic motivators encourage developers to resolve ASAT produced warnings? is investigated. During the posttest questionnaire the participants were asked to answer a set of questions about each game mechanism. Questions were asked about the levels, activity (log and crusher statistics), badges, and leaderboard. Each of these components addressed a combination of the following elements: fun, feedback, challenge, competition, and motivation. The results on these can be found in Chapter 6.

The purpose of this experiment is to test whether developers can be motivated into resolving ASAT produced warnings. Gamification is a way of motivating the developers through various mechanics. Therefore it is important to investigate the extrinsic motivators and their effect on the developers. Each mechanic has been strategically designed and implemented into the dashboard. When a user visits Checkpoint, the first thing visible is the crusher statistics showing how many warnings are resolved. On the side, the latest activities are also visible. The questions addressing the motivators and their scores are found in Section 7.1. The results show that the participants agree that Checkpoint would

motivate them to resolve warnings. The increase in score after the pretest confirms this. Before being exposed to gamification through Checkpoint, the participants were on average less optimistic about gamification.

The fact that participants were not optimistic about gamification is not at all a surprise. As seen in the background study in Chapter 2, gamification within Software engineering is relatively new. During the interview developers were asked whether they think Checkpoint would motivate developers to resolve ASAT warnings. Eight participants said that it would certainly help. The answers of the participants in combination with their reactions to the feedback from Checkpoint during the experiment shows that the majority of the participants were affected by the extrinsic motivators. Many participants' attention was grabbed by the graph on top and the badges. The graph was specifically interesting because it showed them how many warnings they had just resolved.

A few participants also mentioned that developers would end up fixing things that should not be be fixed. These participants were concerned that developers may just resolve warnings for the sake of points and competition. This is a legitimate risk of using gamification which participants have repeated many times. Ironically, in such a situation, the extrinsic motivators might be more effective than desired.

One participants mentioned that the feedback from Checkpoint is a confidence booster because on can visualize their progress. The scores and responses from the participants do indicate that extrinsic motivators have an impact for resolving ASAT warnings.

7.9 Competition

This section investigates **RQ4.2**: Does competition through gamification encourage developers to resolve ASAT produced warnings? Competition can be classified as an extrinsic motivator. The elements which incorporated competition where badges and leaderboard.

Results from the posttest in Chapter 6 show that the participants liked competition. One notable observation during the experiments was that some participants objected against competition at the start of the experiment. However, as the experiment progressed they were triggered by the competitive elements. During this process, the leaderboard was most effective. The leaderboard showed the participants were they stand among other participants. Some of the participants were acquainted and recognized some users on the board. At the end of the experiment, some participants tried to beat the score of at least one other participant on the board. A quote by one of the participants serves this observation quite well: "It is Human nature to compete, whether we like to admit it or not".

Participants were asked about their thoughts on competition through gamification for resolving warnings during the interview. A few participants responded by saying that the competition is a fun thing to have in a team. One participant also mentioned being more focused on their personal progress instead of the progress of others. Another stated that it is fun in bragging rights, but may lose its effect in the future.

The results show that based on the scores, the competition was quite interesting. During the interview most participants were happy with the competition elements, however, some did not like the idea. The criticism here was that competition may end up being bad com-

petition or that it may stop being effective. It is safe to say that based on the responses and scores, on average competition did motivate the developers.

7.10 Most Effective Mechanism

RQ4.3: Which gamification mechanic is most effective in encouraging developers to resolve ASAT produced warnings? The previous questions explored the effectiveness of the motivators. Based on the results, the motivators have an effect on the participants. However, each mechanism has a different effect on a user. The idea behind this question was to explore the effectiveness behind the mechanisms.

Looking at the scores from the posttest, the leaderboard is the number one motivating element in this game. After the leaderboard, the activity mechanism is most effective. The activity log mechanism consists of the log and the graph. Considering that many participants have warned about competition pertaining to serious work, they subconsciously like the competition.

The fact that the activity log mechanism is the second best effective mechanism is a possible explanation that the participants were indeed motivated by feedback. The activity log and stats provided the participants with feedback after each commit. The log showed how many warnings they resolved, what type of warnings they had resolved, and many more game details. Based on the results one could safely say that competition and feedback are two of the most effective mechanisms for motivating developers.

7.11 Effectiveness of Checkpoint

RQ4.4: Is checkpoint effective in motivating developers to resolve ASAT produced warnings? The effectiveness of this tool lies in its ability to stimulate the users such that its goals are achieved. In Section 7.2 a list of questions are asked regarding the effectiveness of Checkpoint. As explained in the section, the questions are asked to measure the possible effectiveness because measuring the actual effectiveness during this experiment is not feasible.

Participants were asked whether they deemed Checkpoint useful during the code review process. The rationale behind this question is that during code review ASAT warnings might be present. If Checkpoint were in place, the teammates would see the warnings and be triggered to resolve these warnings. Some developers might even be more motivated to commit code which is free from ASAT warnings because they know that they will receive positive feedback after doing so.

On average the participants were positive about the effectiveness of Checkpoint. However, based on the interview data, participants are concerned with the effectiveness wearing out over time. One method of keeping the effectiveness up is having periodic leaderboards or maybe incentives from within the company where the tool is applied.

7.12 Usability of Checkpoint

RQ4.5: Is Checkpoint sufficiently usable for developers? The usability of the tool lies in its ability to be operated by an intended user with minimal assistance. During the interview, many participants stated to find Checkpoint intuitive and easy to use. The tool was also called mature and nicely designed.

The results from the posttest in Chapter 6 show that participants needed some help in understanding the purpose of the widgets. This was also observable during the experiments. Many participants were trying really hard to get familiar with the dashboard after registration. The crusher statistics did show a text with an explanation, but that was not effective enough. Many participants did not even read the text. This is an issue which could be fixed with a relatively simple solution in the future, on-boarding. Onboarding is the process used by many applications featuring many functions. During this process, the user is guided to significant elements of an application. Each element is then provided with a short explanatory text.

During the experiment, participants were guided by the observer when things got unclear. As participants started committing code and the dashboard responded with feedback, participants started to get an improved understanding of each widget. The problem was actually an empty dashboard with no on-boarding process.

Overall the tool was easy to use, and the GitHub integration worked seamlessly. The participants did not mind the processing time for each commit either. This was also not a problem because many participants kept going back to their IDE after having made a commit. They only checked the dashboard after a few commits or in some cases at the end of the experiment. This shows that even though notifications were at times quite annoying, the participants were so focused that they were not always nudged by them.

7.13 Threats to Validity

In this section, the limitations of the experiment and its results are discussed. Perry et al. [28] identified three types of threats to validity: construct validity, internal validity, and external validity.

Construct Validity Construct validity measures how well the test measures the construct. The construct here is the hypothesis being tested. In this experiment, the effect of gamification on resolving ASAT warnings is tested. The independent variables as defined in Chapter 5 are motivation, effectiveness, and usability. The experiment has been set up in such a way to simulate a possible real development environment. In this environment, the participant is using Eclipse, GitHub, Checkstyle, and other tools required for the development tasks. The experiment was a pretest-posttest pre-experimental test. In this test, the participants were provided with a pretest questionnaire containing questions about their expectations about gamification. In the posttest, the questions about expectations are repeated as per the design of the test. Participants might be able to guess the desired outcome of the result based on the pretest questionnaire and answer less truthfully.

Another threat is leading questions during the interview. To avoid this, the questions were carefully set up in order to avoid leading.

Internal Validity The internal validity measures how well the experiment was performed. The internal validity is threatened by confounding variables [32] . The degree of control over confounding variables determines the internal validity.

In this experiment, the development environment itself was fully controlled. The setup consisted of tools a developer would normally use during a regular workday. In order to be able to measure the true effectiveness of gamification, the application needs to be put to test for a longer period of time within a real team of developers working on the same project. Such a project would also create other threats and variables which would be beyond the control of a researcher.

A significant part of measuring the effectiveness of gamification is time. The main threat to its effectiveness is the probability that its effect will reduce over time and developers may not be motivated. This effect can only be tested in a longitudinal study. The impact of gamification on the code review process could also only be measured in such an experiment.

The regression towards mean is also a known threat. Extreme scores may result in skewed results. In order to combat this, for each result set the mean and median was computed. These values were compared to see if the difference between the mean and median was significant. None of the result sets resulted in such a difference.

There was one unexpected change during the experiments. A game widget called BugsBuddy was implemented and questions were put into the posttest pertaining to this widget. However, none of the participants had the time to even notice the widget. This resulted in participants not being able to answer questions in the section about this widget. Participants were instructed to just mark something and ignore the section. The results were removed from the experiment to avoid bias. No further analysis was performed on BugsBuddy.

External Validity The external validity is aimed to investigate whether or not the research is applicable to the "outside world". The study can be generalized outside the research environment based on the population validity and development environment validity.

The development environment was replicated as close as possible to a real-world scenario. However, both the population and the environment are not diverse enough to be generalized. This result should not be surprising. In Chapter 5 it was stated that this experiment is a pre-experimental test. In such a test no hard facts can be concluded. The results of such an experiment are a reason for further study. Furthermore, this study may function as a base study for other review studies.

Chapter 8

Conclusions and Future Work

This chapter will draw conclusions based on the results and discussion. Afterward, an overview of the contributions of this project is given. Finally, some ideas for future work will be discussed.

8.1 Conclusions

In Chapter 1 four main research questions were formulated. Based on those questions a literature study was performed. The findings of that study resulted in the implementation of Checkpoint.

- **RQ1:** Why do developers ignore warnings produced by ASATs?
- **RQ2:** What existing gamification models for improving software engineering processes exist?
- **RQ3:** Are developers willing to use gamification within their software development process in order to resolve ASAT produced warnings?
- **RQ4:** Does gamification encourage developers to resolve ASAT produced warnings?

Why developers ignore warnings produces by ASATs The literature study found various reasons why developers would ignore warnings. An overload of warnings and the relevancy of warnings are pitfalls for ignoring warnings. A high number of false positives also contribute to this. Furthermore, when developers do not understand what a warning means, they are not able to fix it. As a result, these warnings stay in the system. The study also found that old bugs have a lower chance of being fixed. The combination of these issues creates a pool of pitfalls which lead to a bunch of unresolved warnings.

Existing gamification models for improving SE The application of gamification to software engineering processes is a relatively new field. The application of gamification to static analysis tools was only tested in one other paper found. The correct application of

gamification to a SE process is significant for its success. The GOAL (Gamification focused On Application Lifecycle Management) methodology was found during this process. The purpose of this methodology is to provide a systematic approach to designing the gamification system. The approach starts off by first determining the objectives of gamification and player analysis. Based on this analysis the scope and game economy is created. The GOAL methodology creates a foundation for the implementation of the actual tool.

Willingness to use gamification within SE The results and discussion show that on average the developers are excited about gamification. Many developers stated to be willing to use gamification within software engineering. At the same time, developers were also concerned about the effects of gamification. The concern was that developers would resolve warnings just for the sake of gaining points. The majority of the developers identified gamification as a fun and useful additions to their daily development process.

Does gamification encourage developers to resolve ASAT produced warnings? To answer these questions, five sub-questions were asked. The results showed that the extrinsic motivators in Checkpoint did motivate the participants. Even though participants were concerned about competition, they did enjoy the competition during the experiment. The leaderboard was even ranked as the most effective mechanism in encouraging developers to resolve warnings. The second most effective was the activity log mechanism. The feedback provided by Checkpoint turned out to be a powerful motivator. Feedback provided the participants with an overview of their progress which in turn boosted their motivation.

On average the participants were positive about the effectiveness of Checkpoint. However, the effectiveness could not be measured during this experiment. The effectiveness was based on the expectation of the participants. Participants were mostly concerned about the motivational effect of gamification wearing off.

Overall, based on the results of these questions it is safe to conclude that gamification did encourage the developers to resolve ASAT produced warnings.

8.2 Contributions

This thesis has shown that gamification has the potential to be used within software engineering. There are pitfalls, but these should be considered and monitored when putting in practice. The work in this thesis makes the following contributions:

- Checkpoint: a gamification tool for keeping track of warnings resolved Checkstyle. The tool shows that gamification can have an effect on the motivation of developers.
- Gamification: this thesis shows another application of gamification within software engineering. The results of this experiment show that the GOAL methodology can be successfully used as a foundation for developing a gamification platform in an unexplored environment.
- Motivators of Gamification: the results of this experiment also show that Feedback and Competition are strong motivators.

8.3 Future work

More work needs to be done in order to find out if gamification is indeed effective in the long run. For the future, a non-pre-experimental study would need to be executed. A quantitative and qualitative study is needed in order to draw hard conclusions. A study with larger population size and more diversity among tools tested would also provide insight on the applicability of such a tool in the "real world".

Another suggestion is the creation of dynamic tasks and achievements. In this project, the tasks and achievements were generated beforehand because the project was known. In practice, such a tool would be used by many different teams and different projects. Each project would be different. It would be a challenge to developer achievements to fit all types of projects with various difficulty levels. An algorithm which could possibly generate interesting tasks based on the current state of the repository would be a significant improvement. That way each project would have their own tasks tailored to their project instead of a generalized achievement set.

Bibliography

[1] Bilal Amir and Paul Ralph. Proposing a theory of gamification effectiveness. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 626–627. ACM, 2014.

[2] Satoshi Arai, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. A gamified tool for motivating developers to remove warnings of bug pattern tools. In *2014 6th International Workshop on Empirical Software Engineering in Practice*, pages 37–42. IEEE, 2014.

[3] Nathaniel Ayewah, William Pugh, J David Morgenthaler, John Penix, and YuQian Zhou. Evaluating static analysis defect warnings on production software. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 1–8. ACM, 2007.

[4] Earl R Babbie. *The basics of social research*. Cengage Learning, 2013.

[5] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. Analyzing the state of static analysis: A large-scale evaluation in open source software. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, volume 1, pages 470–481. IEEE, 2016.

[6] Kay Berkling and Christoph Thomas. Gamification of a software engineering course and a detailed analysis of the factors that lead to it's failure. In *Interactive Collaborative Learning (ICL), 2013 International Conference on*, pages 525–530. IEEE, 2013.

[7] Tim Buckers, Clinton Cao, Michiel Doesburg, Boning Gong, Sunwei Wang, Moritz Beller, and Andy Zaidman. Uav: Warnings from multiple automated static analysis tools at a glance. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 472–476. IEEE, 2017.

[8] Donald T Campbell and Julian C Stanley. Experimental and quasi-experimental designs for research. *Handbook of research on teaching. Chicago, IL: Rand McNally*, 1963.

[9] Gerardo Canfora, Luigi Cerulo, and Massimiliano Di Penta. Identifying changed source code lines from version repositories. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*, pages 14–14. IEEE, 2007.

[10] Tommaso Dal Sasso, Andrea Mocci, Michele Lanza, and Ebrisa Mastrodicasa. How to gamify software engineering. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 261–271. IEEE, 2017.

[11] Michiel De Wit, Andy Zaidman, and Arie Van Deursen. Managing code clones using dynamic change tracking and resolution. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 169–178. IEEE, 2009.

[12] Silviya Dencheva, Christian R Prause, and Wolfgang Prinz. Dynamic self-moderation in a corporate wiki to improve participation and contribution quality. In *ECSCW 2011: Proceedings of the 12th European Conference on Computer Supported Cooperative Work, 24-28 September 2011, Aarhus Denmark*, pages 1–20. Springer, 2011.

[13] Ekwa Duala-Ekoko and Martin P Robillard. Tracking code clones in evolving software. In *29th International Conference on Software Engineering (ICSE'07)*, pages 158–167. IEEE, 2007.

[14] Daniel J Dubois and Giordano Tamburrelli. Understanding gamification mechanisms for software development. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 659–662. ACM, 2013.

[15] Brian J Fogg. A behavior model for persuasive design. In *Proceedings of the 4th international Conference on Persuasive Technology*, page 40. ACM, 2009.

[16] Félix García, Oscar Pedreira, Mario Piattini, Ana Cerdeira-Pena, and Miguel Penabad. A framework for gamification in software engineering. *Journal of Systems and Software*, 132:21–40, 2017.

[17] Todd Helmenstine. What is the difference between independent and dependent variables? `https://www.thoughtco.com/independent-and-dependent-variables-differences-606115`, 2018. Accessed: 2019-01-11.

[18] Yoshiki Higo, Keisuke Hotta, and Shinji Kusumoto. Enhancement of crd-based clone tracking. In *Proceedings of the 2013 International Workshop on Principles of Software Evolution*, pages 28–37. ACM, 2013.

[19] Susan A Jackson and Herbert W Marsh. Development and validation of a scale to measure optimal experience: The flow state scale. *Journal of sport and exercise psychology*, 18(1):17–35, 1996.

[20] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering*, pages 672–681. IEEE Press, 2013.

[21] Sunghun Kim and Michael D Ernst. Which warnings should i fix first? In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 45–54. ACM, 2007.

[22] Gabriela Kiryakova, Nadezhda Angelova, and Lina Yordanova. Gamification in education. In *Proceedings of 9th International Balkan Education and Science Conference*. Proceedings of 9th International Balkan Education and Science Conference, 2014.

[23] Janaki Kumar. Gamification at work: Designing engaging business software. In *International conference of design, user experience, and usability*, pages 528–537. Springer, 2013.

[24] Robyn Longhurst. Semi-structured interviews and focus groups. *Key methods in geography*, pages 117–132, 2003.

[25] Jeanne Nakamura and Mihaly Csikszentmihalyi. The concept of flow. In *Flow and the foundations of positive psychology*, pages 239–263. Springer, 2014.

[26] Sebastiano Panichella, Venera Arnaoudova, Massimiliano Di Penta, and Giuliano Antoniol. Would static analysis tools help developers with code reviews? In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 161–170. IEEE, 2015.

[27] Oscar Pedreira, Félix García, Nieves Brisaboa, and Mario Piattini. Gamification in software engineering–a systematic mapping. *Information and Software Technology*, 57:157–168, 2015.

[28] Dewayne E Perry, Adam A Porter, and Lawrence G Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 345–355. ACM, 2000.

[29] Joseph R Ruthruff, John Penix, J David Morgenthaler, Sebastian Elbaum, and Gregg Rothermel. Predicting accurate and actionable static analysis warnings: an experimental approach. In *Proceedings of the 30th international conference on Software engineering*, pages 341–350. ACM, 2008.

[30] Richard M Ryan and Edward L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist*, 55 (1):68, 2000.

[31] Jan Schumacher, Nico Zazworka, Forrest Shull, Carolyn Seaman, and Michele Shaw. Building empirical support for automated code smell detection. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 8. ACM, 2010.

[32] Marion K Slack and Jolaine R Draugalis Jr. Establishing the internal and external validity of experimental studies. *American Journal of Health-System Pharmacy*, 58 (22):2173–2181, 2001.

[33] Donna Spencer. *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[34] Naomi Unkelos-Shpigel and Irit Hadar. Gamifying software development environments using cognitive principles. In *CAiSE Forum*, pages 9–16, 2015.

[35] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *Social computing (SocialCom), 2013 international conference on*, pages 188–195. IEEE, 2013.

[36] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Andy Zaidman, and Harald C Gall. Context is king: The developer perspective on the usage of static analysis tools. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 38–49. IEEE, 2018.

[37] Yan Zhang and Barbara M Wildemuth. Unstructured interviews. *Applications of social research methods to questions in information and library science*, pages 222–231, 2009.

[38] Jiang Zheng, Laurie Williams, Nachiappan Nagappan, Will Snipes, John P Hudepohl, and Mladen A Vouk. On the value of static analysis for fault detection in software. *IEEE transactions on software engineering*, 32(4):240–253, 2006.

Appendix A

Experiment Information Sheet

Checkpoint

Introduction

In this experiment, you will be working with JPacman a project developed at the Technical University of Delft with the purpose of teaching student the art of testing and software quality, and Checkstyle. Checkstyle is an automated static analysis tool (ASAT) which helps programmers write Java code that adheres to a coding standard. Coding standards or conventions are a set of rules which recommend a programming style such as method naming, spacing, comments, etc. There are many ASATs out there for various languages, however, for this project you will be working with Java and Checkstyle. The experiment should not take more than an hour in total. For the programming part of this experiment, you have 20 minutes.

JPacman

JPacman is inspired by Pacman and written in Java. The purpose of JPacman is to teach students the art of testing. In case you are a TU Delft student, chances are, you are familiar with JPacman. The code version you will be working on is incomplete and missing some functionality. You can run JPacman by executing MainUI.java. After a few seconds, the Pacman game should be visible for you to test out.

Checkpoint

Checkpoint is a tool which has been developed as part of this experiment. In Checkpoint, you will be able to visually keep track of the number of warnings you have resolved over time. By resolving warnings you will be earning XP points. There is also a set of badges to earn after completing various tasks. The badges also reward you with XP points. In the leaderboard, you are able to see how you compare against other players. The more warnings you resolve, the more XP you gain, and the cleaner your code becomes.

Development Environment

Your development environment is already set up for you. You will be working in Eclipse for Java. The Checkstyle plugin is already installed and configured for the project. The source code is hosted on GitHub. Within the local GitHub application, you will be able to commit and push your changes to GitHub. In your browser, you will register and login into Checkpoint. You will first need to navigate to the Projects page in Checkpoint and join the project which has been assigned to you.

A few things to look out for

Checkpoint is still a prototype and therefore the queueing system of the computation engine is not functional yet. What this means is, that you need to wait at least 30 seconds between a push. You may push multiple commits at once, but make sure to wait a minute for the next push. You can just keep working on your main branch, you do not need to create new branches.

Tasks

Let's take a look at what you will do. Read this page in its entirety first.

Registration

First, you will need to go to Checkpoint and register your Account by filling in the form after clicking "Register a new membership". Make sure to register using the following GitHub username: **Raies1000**. That is the username for the commits which will take place.

Fun with Programming

Your main task for this assignment is to resolve warnings produced by Checkstyle. The warnings can be found in the Problems tab of your Eclipse IDE. You are free to choose which warnings and how many warnings you resolve.

Your checklist:

- Register in Checkpoint
- Join the project assigned to you
- Get familiar with Checkpoint
- Open Eclipse and get familiar with JPacman
- Get JPacman to run
- **Resolve warnings produced by Checkstyle. Try to resolve the warnings as you would normally.**
- **After resolving a few warnings, commit and push your code via the GitHub application.**
- Navigate the Checkpoint dashboard during this process.
- Try to unlock a few badges by completing the tasks viewed by the badge.
- If you find a friend or colleague on the leaderboard, try to beat them at their score

Repeat the tasks in bold.

Feel free to ask any questions now or during the experiment!

Head to Firefox (the blue icon) to start Checkpoint.

Good Luck and Have Fun!

Appendix B

Experiment Procedure Sheet

Checkpoint

Welcome (5 min)

Welcome the participant

About the Experiment sheet + Pretest (Form B) (10 min)

Show PDF, printout

https://goo.gl/forms/G65OT64fw6rsc6412

Introduction + Programming (20 min)

The participant will program

Posttest (Form A) (10 min)

https://goo.gl/forms/CqdcdKNCPVflTH6N2

Interview (15 min)

Interview the participant

My Checklist

- Make sure webhook redirect for GitHub is active
- Couple [redacted] with personal Checkpoint account
- Create Repo on GitHub and checkout
- Add Clean JPacman code and push to GitHub
- Verify if processing by Checkpoint has completed OK
- Change personal GitHub username in Checkpoint to *fake*

- Open new Desktop for experiment
  - Have Eclipse ready for project
  - Have GitHub ready
  - Have Checkpoint ready

- Register new user with GitHub username [redacted]
- Join user in project
- Let the fun begin

- After the experiment, switch GitHub username back.

Appendix C

Pretest Questionnaire

Pretest Form (B)

Thank you for participating in this experiment.

* Required

1. **Your age?** *

2. **What is your educational background?** *

3. **What is your current occupation?** *

Experience

4. **How many years of programming experience do you have?** *

5. **I am experienced in Java Development** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

6. **I work in teams when developing software** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

7. **I am familiar with Eclipse for development** *
Mark only one oval.

- ◯ Yes
- ◯ No

About Software Quality

8. **Functional code is more important than clean code** *
Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Completely Agree

9. **When programming, I write documentation for all code** *
Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Completely Agree

10. **When programming, I write tests for all code** *
Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Completely Agree

11. **There is a code review process in the development process of the projects I work on** *
Choose "Yes" if one or more of the projects you work on have a code review process
Mark only one oval.

- ◯ Yes
- ◯ No

12. **I find the code review process useful**
You can skip this question if you answered "No" to the previous one.
Mark only one oval.

	1	2	3	4	5	
Totally disagree	◯	◯	◯	◯	◯	Completely Agree

13. **There are coding conventions in place for the projects I work on** *
Choose "Yes" if one or more of the project of the projects you work on have coding conventions in place.
Mark only one oval.

- ◯ Yes
- ◯ No

14. **I obey to the coding conventions when programming**
You can skip this question if you answered "No" to the previous one.
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

On ASATs

Automated static analysis tools perform a static analysis on the code providing warnings for functional or maintainability problems. Some examples of ASATs are: Checkstyle, PMD, FindBugs, JSLint. In this section you are asked a series of questions about your experience with ASATs.

15. **Automated Static Analysis tools (ASATs) contribute in upholding the quality of the code I write** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

16. **Warnings generated by ASATs are completely useless** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

17. **How many Static Analysis tools do you use at one given time?** *
Mark only one oval.

- ◯ None
- ◯ One
- ◯ Two
- ◯ Three or more

18. **I use Automated Static Analysis tools to inspect my code** *
Mark only one oval.

- ◯ During development
- ◯ During code review process
- ◯ Before commiting
- ◯ I never use it

19. **When an ASAT generates warnings I tend to** *
Mark only one oval.

- ◯ Completely Ignore the warnings
- ◯ Only resolve warnings I consider important
- ◯ Resolve warnings pertaining to my code
- ◯ Resolve warnings pertaining to my code and from commits of others
- ◯ Resolve warnings only if the CI fails
- ◯ Resolve all warnings

Expectations

This section is about your expectations for a tool which would enhance developer motivation for tedious tasks.
This tool would gamify part of the development process.

By gamifying the use of static analysis tools one should be able to have more fun in resolving tedious warnings. The goal of gamification is to make tasks (especially tedious tasks in serious

environments) more fun. Resolving warnings should award the user with experience, badges and a ranking among friends or colleagues. By adding a gamification tool into the development process developers should be more inclined to perform less exciting tasks.

20. **Gamification would help me resolve more warnings generated by Automated Static Analysis Tools (ASAT)** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

21. **The competition with colleagues will motivate me to resolve more warnings produced by ASATs** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

22. **Such a tool would be valuable to the development process** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

23. **Such a tool would be interfering in my development process** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

24. **Such a tool would ensure that I get less warnings from my colleagues** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

25. **I expect such a tool to motivate my colleagues to commit code free from ASAT warnings** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

26. **I expect ASAT warnings to be taken into account during the code review process** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

27. **I do not see the value in using such a tool** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Appendix D

Posttest Questionnaire

Form (A)

This is the final questionnaire and should not take more than 7 minutes.

* Required

Experiment Experience

1. **The assignment was difficult for me** *
   Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

2. **I felt a lot of time pressure** *
   Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

3. **The experiment was interesting** *
   Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

4. **The guidance for the experiment was enough** *
   Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

UI Experience

In this section you will be asked a series of questions regarding the User Interface of Checkpoint, the tool developed as part of this experiment

5. **I like the interface design of Checkpoint** *
   Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

6. **Checkpoint is easy to use** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

7. **Joining a project in Checkpoint was easy** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

8. **I needed little help in understanding the purpose of the widgets in Checkpoint** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Process Experience

9. **I opened Checkpoint:** *
Mark only one oval.

- ◯ After every commit
- ◯ Occasionally
- ◯ Only when I recieved notification in windows
- ◯ Never
- ◯ Other: _____

10. **Checkpoint as a separate program was inconvenient** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

11. **The fact that checkpoint only works with repositories hosted on GitHub is limiting** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Levels

12. **The level progress bar motivated me to keep going for the next level** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

13. **The level naming was interesting** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

14. **I knew why I was promoted to the next level** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

15. **How easy or difficult did you find to progress to the next level** *
Mark only one oval.

	1	2	3	4	5	
Very Easy	◯	◯	◯	◯	◯	Extremely Difficult

Notifications

16. **The push notifications from Checkpoint were a disturbance to my workflow** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

17. **The push notifications from Checkpoint awakened my curiosity** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

18. **The feedback from Checkpoint through notifications was frequent enough for me** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Activity

19. **The Activity log is a useful widget, it provides me information about my progress** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

20. **The activity log was detailed enough** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

21. **The Crusher Statistics (warnings resolved chart) is a useful widget as it provided me information about my progress** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

22. **If I could see how many warnings in a day other users resolved, my motivation to resolve more would increase** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Badges

23. **Badge Achievements motivated me to resolve more warnings in order to unlock them** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

24. **The tasks to unlock a badge could have been more interesting** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

25. **The badge naming was creative** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

26. **The task progress of badges were clear** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

27. **How easy or difficult did you find the tasks to unlock badges** *
Mark only one oval.

	1	2	3	4	5	
Very Easy	◯	◯	◯	◯	◯	Extremely Difficult

28. **Seeing how many users unlocked a badge motivated me to earn that badge** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

BugsBuddy

29. **BugsBuddy motivated me to keep resolving warnings** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

30. **When BugsBuddy was sad, I was more inclined to resolve warnings** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

31. **I found the idea of having a virtual buddy creative** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

32. **The expressions of BugsBuddy should have been better** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Leaderboard

33. **If found it useful to see were other players rank in the game** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

34. **The leaderboard motivated me to gain XP** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

35. **If I had a leaderboard with my friends, it would motivate me more than having just a global leaderboard** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Checkpoint

36. **Checkpoint motivates me to resolve more warnings than I usually do** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

37. **The competition with colleagues will motivate me to resolve more warnings produced by ASATs** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

38. **Checkpoint would be a valuable tool to my development process** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

39. **Checkpoint would be interfering in my development process** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

40. **Checkpoint would ensure that I get less warnings from my colleagues** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

41. **I expect Checkpoint to motivate my colleagues to commit code free from ASAT warnings** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

42. **I expect ASAT warnings to be taken into account during the code review process when Checkpoint is used** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

43. **I do not see the value in using Checkpoint** *
Mark only one oval.

	1	2	3	4	5	
Totally Disagree	◯	◯	◯	◯	◯	Completely Agree

Experiment Rating

44. **The clearness of the assignment** *
Mark only one oval.

	1	2	3	4	5	
Very bad	◯	◯	◯	◯	◯	Excellent

45. **The quality of the selected project JPacman** *
Mark only one oval.

	1	2	3	4	5	
Very bad	◯	◯	◯	◯	◯	Excellent

46. **The quality of the information document** *
Mark only one oval.

	1	2	3	4	5	
Very bad	◯	◯	◯	◯	◯	Excellent

47. **The quality of the questionnaire** *
Mark only one oval.

	1	2	3	4	5	
Very bad	◯	◯	◯	◯	◯	Excellent

48. **The opportunity to give feedback** *
Mark only one oval.

	1	2	3	4	5	
Very bad	◯	◯	◯	◯	◯	Excellent

49. **Overall impression** *
Mark only one oval.

	1	2	3	4	5	
Very bad	◯	◯	◯	◯	◯	Excellent

Appendix E

Checkpoint

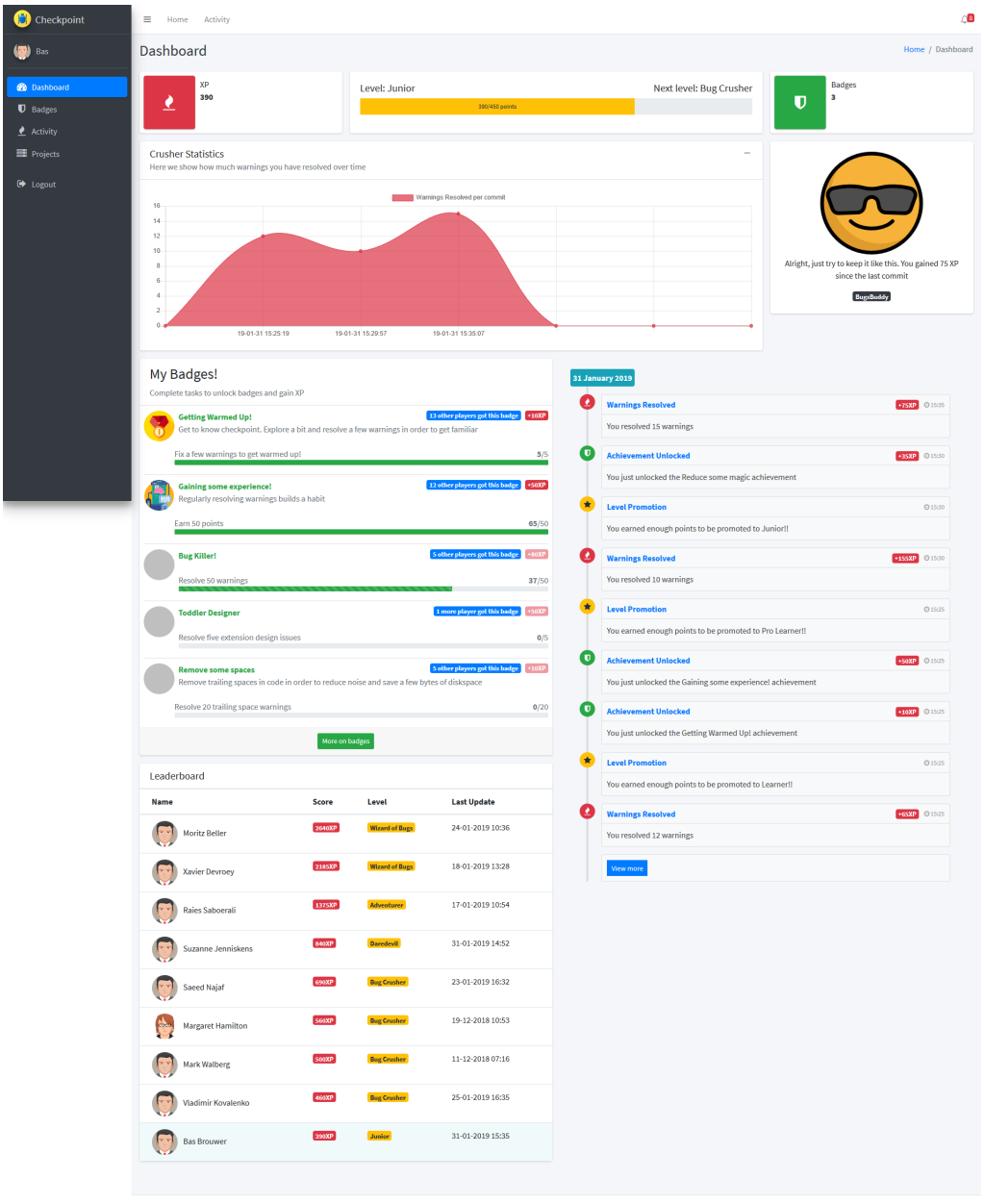

Figure E.1: Checkpoint Dashboard

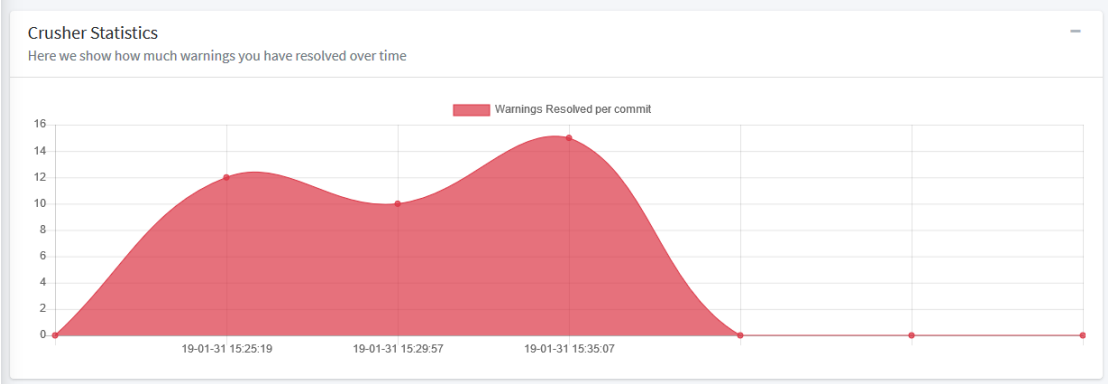

Figure E.2: Checkpoint Crusher Statistics

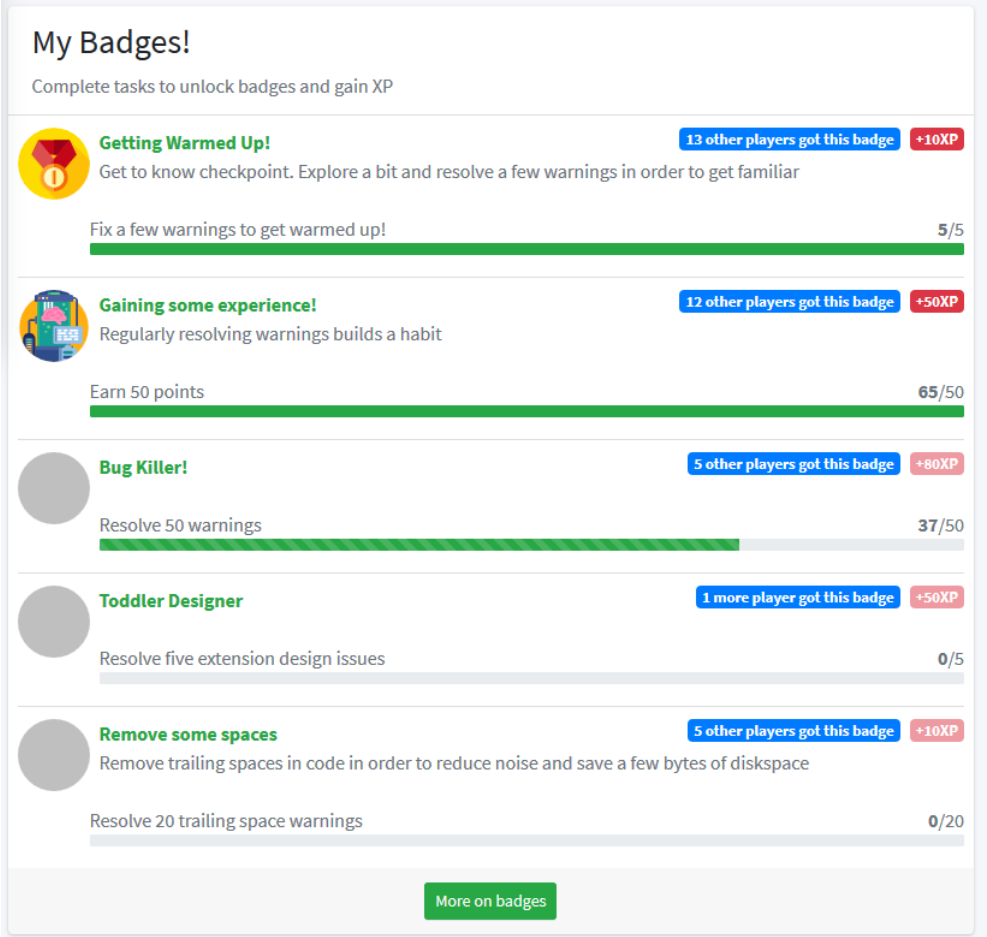

Figure E.3: Checkpoint Badges

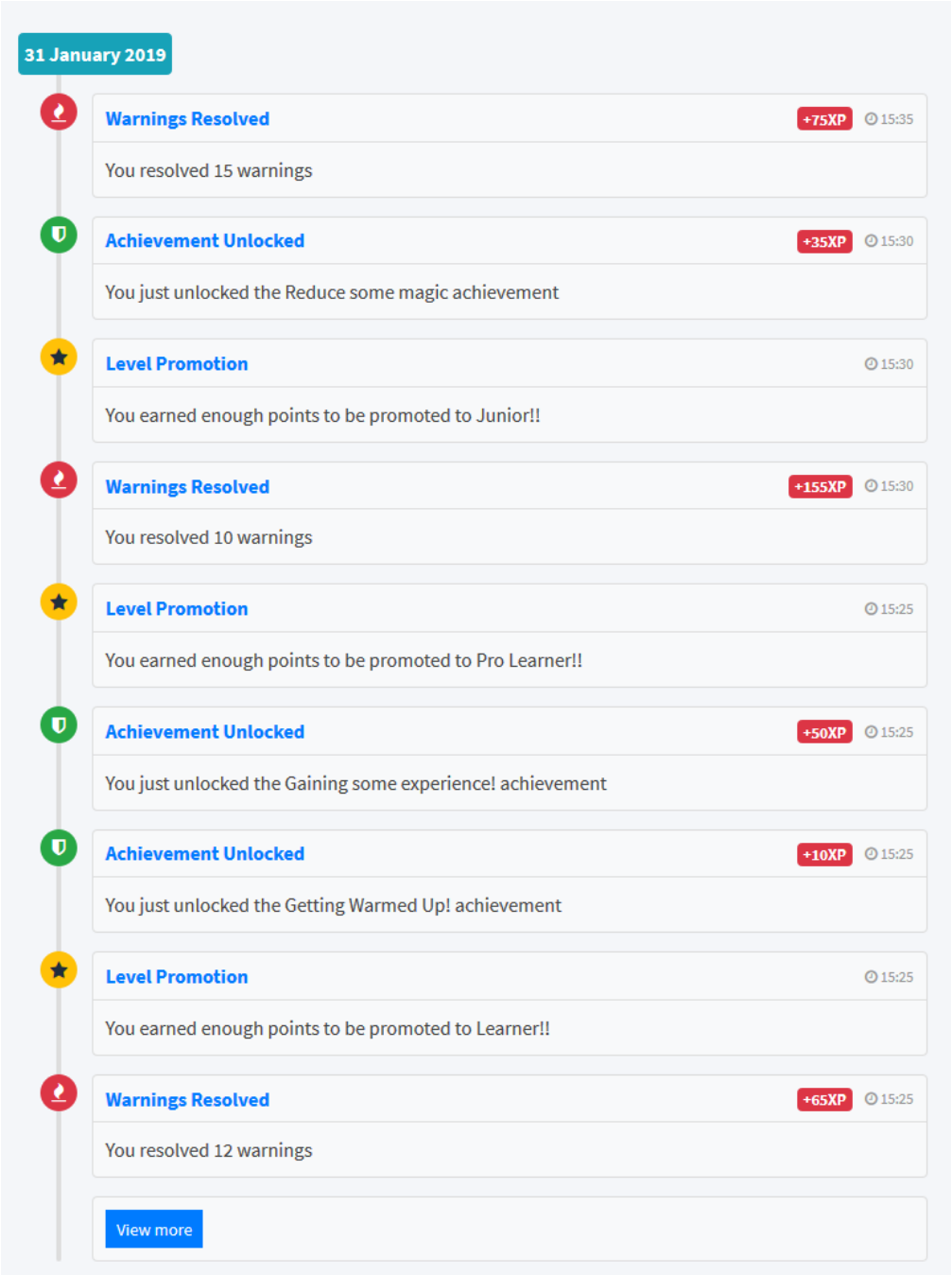

Figure E.4: Checkpoint Activity Log

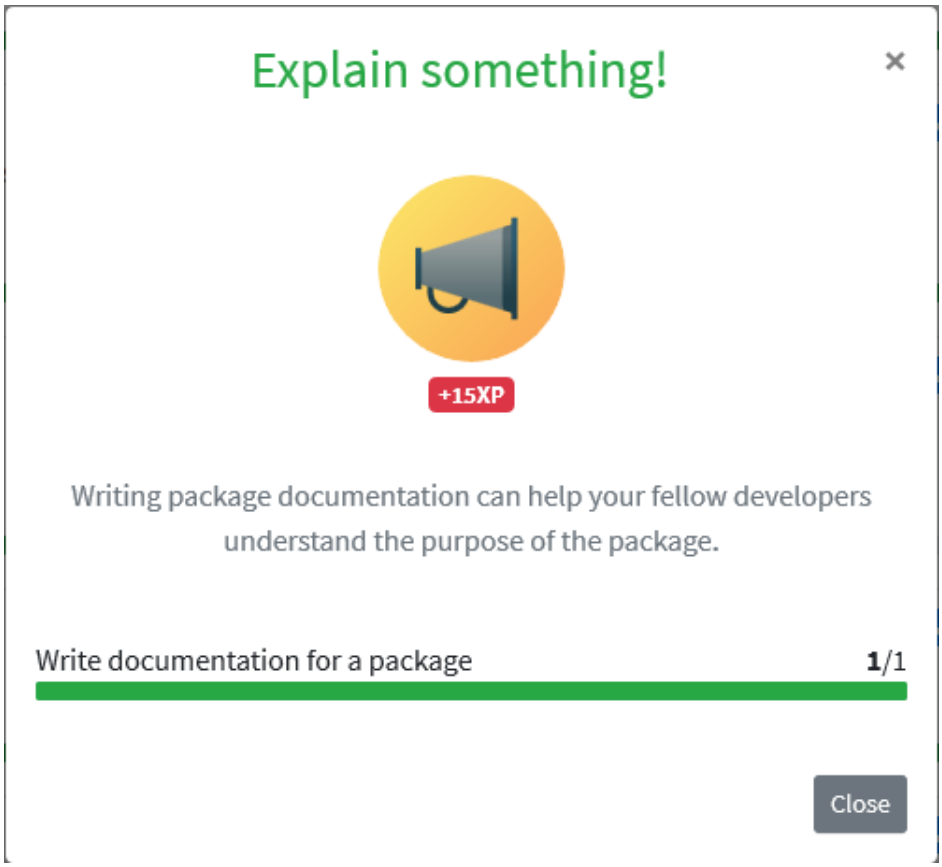

Figure E.5: Checkpoint Badge Example

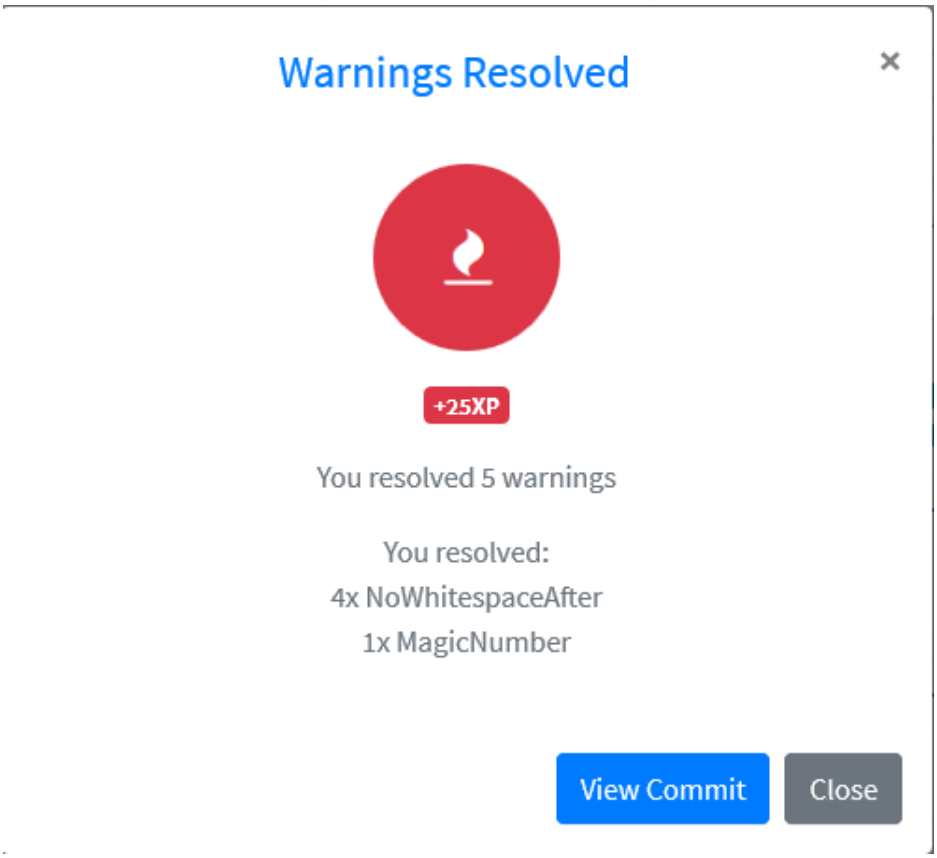

Figure E.6: Checkpoint Activity Example