CHALMERS

Improving Unit Testing Practices With the Use of Gamification

Masters Thesis in Software Engineering

DAVÍÐ ARNARSSON
ÍVAR HÚNI JÓHANNESSON

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Improving Unit Testing Practices
With The Use Of Gamification

DAVÍÐ ARNARSSON
ÍVAR HÚNI JÓHANNESSON

Supervisor: ERIC KNAUSS
Examiner: RICHARD BERNTSSON SVENSSON

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden, June 2015

Abstract

Background Testing in software development is important. Unit testing is one technique developers use to catch potential bugs and combat regressions. However, unit testing is often considered to be an arduous task, and as such, developers might avoid unit testing their code. Gamification is a concept that has been defined as the use of game elements in a non-game context. The concept has been demonstrated to have a positive effect on unit testing in an experimental context but hasn't been observed in industry until recently.

Aim This study employs gamification as a means for motivating developers to write more and better unit tests. By analyzing both static and dynamic qualities of unit tests and feeding the results into our gamification tool, G-Unit, we aim to score developers based on their unit testing efforts.

Method This study applies design science research, based on work by Wieringa. The study consisted of 3 iterations and each iteration consists of 5 phases of the regulative cycle. The data was collected by using surveys, interviews and by mining the artifact's SQL database.

Results No ethical issues arose during the study, the developers felt that the G-Unit tool motivated them to write more and better tests and were able to learn about important software testing metrics and concepts. The largest design problem was the continuous balancing of the gamification system while the largest technical challenge was the implementation of test smell detection.

Contributions The study was a successful in its task of motivating the developers at the participating company to write more and better unit tests. Gamification had a positive effect on the developers work.

Keywords: Gamification, Unit-testing, motivation, testing, ethics, design science research.

Acknowledgements

We would like to thank Eric Knauss, our supervisor, for giving us invaluable feedback and guidance, and Emil Alégroth for introducing us to test smells. We would like to thank Advania, for their participation and continued interest in our work. We would also like to thank our classmates, Þorvaldur Gautsson and Jacob Larsson for their proof-reading. Finally, our thanks go to Arna Sif Ásgeirsdóttir, for her support, and the many delicious meals she cooked for us.

The authors, Gothenburg, 24. May 2015

Contents

1 Introduction

Software testing is an important practice in software development. Testing can be used to increase a tester's confidence in his code, uncover faults and verify up to a degree that the system meets its requirements. Although testing is a vital part of software development, testing cannot guarantee the absence of faults no matter how creative or well designed the tests are [1]. There are various types of testing, which can be done either manually or automatically through the use of tools. Unit testing, the testing method studied in this thesis, is a commonly used type of automated testing. It focuses on testing individual units of source code, and is an important component of a number of Agile development methodologies and practices, such as Test-Driven Development, and eXtreme Programming.

Testing software has both quality and cost benefits. Envisioning the benefits is easier from a quality perspective than the cost perspective since one of the primary goals of software testing is to uncover faults. Maximilien [2] demonstrated that not using a methodology which includes testing as a first-class citizen can severely affect project quality in the form of defects. For instance, he demonstrated a 40% decrease in defects between a software project which employed test-driven development compared to a baseline project which employed ad-hoc testing.

The idea that testing can cut development costs might sound somewhat strange. However the scientific literature on the subjects seems to adhere to that statement. Maximilien

[2] argues that one of the most effective ways to keep development costs down is to minimize the number of defects. K. Briski et al. [3] argue that since the cost of catching a defect grows exponentially as the software progresses through the development cycle it is vital to catch defects early on. Maximilien [2] suggests that finding a defect after the release of the actual product is up to 30 times more expensive than finding the defect in the design or architectural phase of the product. McConnell, based on a number of studies, suggests that finding a defect post-release might actually cost up to 100 times more than during the requirements phase [4]. The literature mentioned above seems to indicate that writing automated tests results in higher quality software with fewer defects which in turn could lead to lower overall development costs.

Test code can contain flaws just like source code. A. van Deursen et al.[5] defined potential trouble within test code as "test smells", building on top of Martin Fowler's work on code smells [6] and defined a list of test smells and how their effects could be mitigated through refactoring. Absence of test smells does however not guarantee a well written test. A test has to be carefully written and designed in order to uncover faults within the source code [1].

Keeping developers motivated is important. In the 21st century a new concept, gamification, has been gaining more attention from the software community [7]. For instance, researchers have attributed the success of the mobile application FourSquare to its gamification elements [8]. The definition of the gamification concept varies between researchers, as many similar definitions exist. In this thesis, the definition of gamification given by Deterding et al. will be used.

"The use of game design elements in a non-game context."[8]

Gamification has mostly been studied in the context of education and learning where it has shown good results in regards to motivation, learning and increased enjoyment of the gamified tasks [9]. Johansson and Ivarsson's experiment [10] on the gamification of unit testing also produced positive results where the subjects felt that the testing phase was more fun and interesting compared to a control group. Because of these promising results, this study incorporates gamification into the unit testing practices in order to motivate and increase enjoyment of unit testing at Advania, the industry partner of the study.

1.1 Case company

Advania is a nordic IT company which services a wide array of different types of customers, be it government contracts or businesses. The participating department within Advania manages an educational software called Inna. Inna is used to manage communication and day to day school operations. It is used by students, teachers and support staff at over 30 schools in Iceland, totalling an end-user base of over 40.000 people. The department has just recently released a new version of Inna, which was a total rewrite of their front end, and they are working towards reaching feature parity with the older version. The department also used the release as an opportunity to switch to weekly releases. Their current testing practices are ad-hoc, there are very few if any automated tests, and their testing is performed manually.

1.2 Purpose of the Study

The purpose of the study is to improve software testing at Advania in the context of unit testing by creating a gamification tool which can:

- Analyze code coverage of the unit test
- Detect test smells in the unit test
- Rank the test based on test smells and code coverage
- Increase motivation among the developers to write more unit tests.
- Increase the enjoyment of unit testing among the developers.
- Motivate the developers to write high quality unit tests in terms of code coverage and test smells

By detecting test smells and analyzing code coverage the tool shall be able to rate the tests on a numeric scale. Malayia et al. argues that "[...] modules covered more thoroughly during unit testing are much less likely to contain errors" [11] and therefore the ranking system of the tests will be based partly on code coverage. The test analyzing tool built during this study, named G-Unit, uses JaCoCo which is a code coverage analysis tool that provides detailed statistical info regarding (but not limited to) line coverage, branch coverage and instruction coverage. Those statistics could be interesting to the developers and could indicate the quality of the tests, although some of the statistics will not be used in the gamification system. A brief informal interview with representatives of two different software companies in Iceland [12] [13], other than Advania, revealed that a certain level of code coverage is expected, although actual coverage percentage and which code coverage metric is inspected seems to vary. Including code coverage metrics in the gamification system therefore seems logical [12][13]. By rating the tests based on test smells and code coverage data it is possible to create a gamification system that uses the data from the tool as input. As previously stated, gamification has been shown to positively influence the motivation, engagement and enjoyment of the gamified task at hand. With that in mind, the purpose of the gamification system should be to increase the motivation and engagement of unit testing. Hopefully this work will lead to better testing practices at Advania, resulting in higher quality software products and cost reductions through more adequate testing infrastructure.

1.3 Statement of the Problem

After the initial meetings with management at Advania it became apparent that they were facing problems with their testing infrastructure. They had written few unit tests and therefore lacked experience in writing them in the participating department. Their interest and motivation to write tests was also rather low. It became clear after the initial literature review that these problems were not limited to Advania as an organization, but were also applicable to other software companies within the industry. Kanij et al. [14] for instance suggests that testing is often considered to be a boring task within organizations. By incorporating gamification into the testing practices at Advania the study aims to increase the motivation to test, hopefully resulting in more tests. The goal of the study was also to encourage developers to write better tests by constructing a gamification system which encourages the writing of high quality tests by encouraging high code coverage and low amount of test smells. Although many researchers claim that it is important to change the perception of testing within software organizations, the literature review didn't reveal any innovative methods to change the developers' view of testing. Therefore this study does not focus solely on the technical aspects of testing but also tries to improve the social aspect as well.

A potential problem could also be that the developers simply aren't knowledgeable enough about test smells or testing in general. Therefore the developers must be educated alongside the deployment of the tool on what constitutes as a test smell and how to avoid them in order to benefit from this study in the long run.

1.4 Research Questions

RQ1: How does gamification influence unit testing practices?

> **RQ1.1:** How does gamification affect the developers' motivation to write more unit tests?
>
> **RQ1.2:** How do test smells and code coverage as input to a gamification system influence unit test quality?

RQ2: What gamification element in this study proves to be the most effective at influencing unit testing practices?

1.5 Scope and Limitations

After the meetings with Advania it became apparent that they were interested in improving their all around test infrastructure. However the scope of improving all aspects of testing was deemed too broad for a master thesis and the focus was set on unit testing. This is also the field of testing the authors were most interested in and also had some experience in that particular field. After further discussions about unit testing with management it became clear that their problem with testing was not strictly bound to the technical aspect but also in terms of motivation, general knowledge of testing and interest in testing among the developers. As discussed in Section 1.3 companies around the world have acknowledged that their issues with testing are not strictly bound to the technical aspect and therefore including the social aspect of testing within the scope was deemed acceptable.

1.6 Contributions

As an exploratory study, the authors contribute to academia by exploring the possibilities relating to gamification of testing and how Ivarsson and Johansson's [10] findings on gamification of testing work in the real world. Advania is the biggest beneficiary here as it stands to gain something in terms of software quality and cost reductions if the study is successful. Even though the gamification part of the study fails, the company will still benefit from the study. Their developers gain experience in terms of unit testing and will know how to avoid test smells and how to mitigate their effects. Since G-Unit provides an automatic evaluation of their tests they could potentially use the tool without the gamification engine in order to evaluate their tests. The potential audience will likely be practitioners interested in improving their testing infrastructure and academic researchers interested in application of gamification in the real world.

2

Background

The following section provides a background for the important concepts in this thesis. It discusses gamification in terms of benefits, prior work linked with software engineering and ethical issues surrounding gamification. It also provides background for software testing in general, unit testing and test smells within unit tests.

2.1 Gamification

In the last decades of the 20th century the popularity of video games rose substantially and today video games are one of the most powerful entertainment platforms [15] [16]. After the rise in popularity of video games, researchers have become more interested in video games and what makes them appealing. Video games expose the player to certain game mechanics which the player is forced to learn in order to master the game itself. Video games can often spark an interest in non-gaming related topics and therefore researchers have been interested in how to apply video games in an educational context. In recent years, researchers have conducted several studies on this subject and have had interesting results. However, researchers have at times run into problems that outweigh the benefits of applying video games in educational context, for instance related to technical infrastructure. Due to these issues, some researchers have instead focused on applying the positive elements from video games to a non-gaming context [15], also known as gamification.

The term itself, gamification, is relatively new as a research topic within the field of software engineering and in industry. According to Deterding et al [8] the term itself originated within the digital media industry in 2008 but the use of the actual term itself didn't become popular until around 2010. Other sources claim that the term was coined by Nick Peilling in 2002 [17]. Although the age of the term itself is not of high importance, it is safe to say that it originated in the 2000s and became widespread around the year 2010. Although some research has been done on gamification by scholars, there is more to explore with regards to, for example, unit testing, which has on its own been studied quite thoroughly by researchers.

Deterding et al. [8] describe the most common elements of gamification as points, also known as experience points or XP, badges and levels. Most systems give users points/XP for every task that they finish, how insignificant it might be although bigger tasks yield more points/XP. Users earn badges by completing various milestones or quests within the gamified system, for example by completing a task within a certain time interval. A level up system is also quite common. Level up systems revolve around the idea that a person will level up when he or she has gathered enough experience points to advance to the next level. Level up systems are often based on an exponential scale instead of a linear progression meaning that reaching Level 3 is harder (requires more points/XP) than reaching level 2. Another element is a leaderboard, which visualizes the status of the people involved in the gamified task. A leaderboard can include some or all of the following elements: points/XP, badges, milestones/quests and any other statistical info that is gathered throughout the gamified task. Instant feedback is yet another common element in gamification. In a digital environment instant feedback may be easier to implement relative to a non-digital environment, for example by displaying messages when a user acts within the system or with status updates notifying the user of a change in the state of the system.

2.1.1 Benefits of Gamification

Various benefits can potentially be acquired from gamification if applied correctly. Some researchers have reported positive results in regards to motivation, learning and increased enjoyment of the gamified task at hand [9]. Another interesting topic is to influence user behaviour through gamification, e.g. steering the user's behaviour in a certain direction through various gamification elements, for instance through badges.

A good example of badges influencing user behaviour is a study conducted by Grant & Betts. Grant & Betts [18] researched public data from the Q&A website StackOverflow[1] which uses points and badges in order to try to influence users. An example of the manipulation of users behavior is demonstrated with the Copy Editor badge which is awarded to users for editing over 500 posts on the site. The policy of the website is that the content on the website belongs to the community itself and not the user and therefore the website encourages community editing in order to increase the quality of posts on the website. The study illustrates how users that are close to 500 edits actively edit posts until they reach the 500 edit mark and after that significant behavioural changes occur. The users stop editing posts after the Copy Editor badge is acquired as the users feel no need to edit other peoples' work any longer. This demonstrates how the site is able to manipulate users to contribute in form of edits as an increase in edits is observed before the acquisition of the badge compared to the time period after the badge is awarded to the user.

2.1.2 Gamification of Software Development

Researchers have conducted studies on the gamification of software development both in an educational context and in industry and achieved positive results. Singer and Schneider [19] were able to motivate students to commit their changes to a project repository more frequently by using a leaderboard (along with other tools). After the study ended they noted that some students felt slightly uncomfortable with the competitive elements that the leaderboard introduces. However, they noted that although some students felt uncomfortable because of the leaderboard, it seemed to be motivating for students to increase their commit count.

Passos et al. [20] conducted a case study with a real world agile software team. In the study, developers and teams were awarded medals for their completion of tasks, number of iterations completed within a time interval, and medals were awarded for levels of test code coverage. The results indicated that the use of achievements not only motivated the

[1]http://stackoverflow.com

developers to become more engaged in their work but also helped the company monitor and control the development process.

2.1.3 Gamification & Ethics

Implementing a gamification system for use by a real world software development team can be quite a challenge from the ethical perspective. Shahri et al. [21] discuss this in their study on the ethics of gamification. Although Shari's study was not based on software development teams but rather general teamwork, the ethical guidelines the study lays out are still applicable to a software development team since software development is based on team work.

The collaborative nature of a software development team could possibly be an issue, as unnecessary tension could be created between developers with the introduction of a gamification system. This is largely related to the personalities of the developers that make up the team. Employees could potentially become more arrogant when they are awarded achievements, while other may become more helpful and might begin to help their teammates unlock the very same achievements that they earned themselves [21].

Employees that are listed in a leaderboard could feel more reassured about their performance, while others could become depressed about never appearing on the leaderboard at all. This is related to the management style of the team, as managers who use the leaderboard to compare employees could create tension between the employees by ranking them based on their performance on the leaderboard. Employees sometimes prefer a gamification system which would be used to help them improve, without comparing themselves to other employees [21].

Employees tend to be more accepting towards a non-comparing gamification system, since they do not feel as pressured as when the manager uses it to compare performances. Another aspect is that although a gamification system can often easily measure how much work has been done by an employee, the system has difficulties evaluating the quality of the work [21]. To counter that, this thesis' gamification system tries to measure the quality of tests written based on the metrics discussed in section 1.2. However, that is one of the largest challenges of the gamification system and will be discussed further in the validity threat section.

Another scenario is when employees within the same team start to team up within the team. This could either be useful or harmful for the group. Members who are dragging the ‘team score’ down by their bad performance could feel pressured to leave the team since they aren’t contributing enough to the team’s gamification profile. The positive side is when team members with a similar skill set start teaming up [21]. This could perhaps result in more effective teamwork.

Since gamification systems are based on statistical info about its users there are large quantities of personal information that could potentially be gathered. For instance, logging of a person’s work hours, webcam analysis, effectiveness of a person’s work hours and other kinds of personal data [21]. In this study only statistical information about the unit tests written by the employee is recorded and no other information. Employees are often not happy if personal data is widely available and visible to everybody within the company and would rather have the data only visible to their supervisor, since it is often the supervisor’s job to monitor his employees. Employees also deem it more suitable that their statistics are available to their team members and managers rather than other workers outside the team. Again, this is a problem which heavily relies on the employee’s personality since hard working, competitive employees would like to use the logs from their personal statistics as an advantage when they would apply for bonuses or promotions. Low scoring employees that are perhaps not as competitive might have lower scores in the system since they do not care as much about it. Therefore their lower score could be related to their lack of competitiveness instead of the quality and/or quantity of their work [21].

In order to make the goals, the rules of the leaderboard and the gamification system in general more clear, management should clearly state the objective of the gamified tasks [21]. In other words, transparency is key. If the objective is to rank employees and base salary bonuses on the leaderboard, management has to make sure that this is clear to every employee. The objectives of the gamification system in this study is not to create a system for management to rank or evaluate the performance of their developers programming capabilities but only to motivate the developers to write more and better unit tests.

2.2 Maven

Maven is a software build and integration tool for the Java environment. Maven has powerful software dependency management capabilities, as well as a plugin API for writing software which interacts with any aspect of the build phase that Maven is concerned with. For instance, the Maven Surefire Plugin[2] generates reports based on the Maven test phase results. The dependency management functionality revolves around repositories for available software built using Maven. Other software projects using Maven can then depend on this available software down to the exact version in their Maven configuration, and Maven automatically resolves and downloads said dependencies [22].

2.3 Software Testing

G.J. Myers and C. Sandler describe software testing as "[...] the process of executing a program with the intent of finding errors" [1]. Although software testing is a proven method to find bugs and errors within a software suite, many developers have a bad attitude towards it. For instance, both Martin et al. [23] and Kanji [14] et al. note that developers often find the task of testing their software dull, boring or the least fun task in their work. Others describe testing as difficult, time consuming and inadequate. This is quite interesting, especially considering that in 2002 the cost of inadequate infrastructure for software testing is estimated from $22.2-59.5 billion dollars in the United States [24]. Although these figures are quite old, they give insight into the problems that the software industry faces in the context of testing. Although this bad attitude could possibly be a cause for the less than optimal state of software testing in the industry today, more studies are needed to link the attitude towards testing to the state of testing procedures.

Although testing suffers from social problems, technical problems are also present. The knowledge used in software testing suffers from low maturity compared to other engineering practices. Test developers do not base their decisions on facts and undisputed statements but rather on their own intuition, trends and market speak, resulting in unpredictable results [25].

[2]http://maven.apache.org/surefire/maven-surefire-plugin/

2.3.1 Automated Testing

Testing software by hand, also known as manual testing, can be very time-consuming. Automated testing is any form of testing which uses software to automatically execute a set of test scripts, and report the result. Doing so only requires the developer to write the test scripts.

2.3.2 Unit Testing

Whittaker defines unit testing as: "unit testing tests individual software components or a collection of components. Testers define the input domain for the units in question and ignore the rest of the system." [26]. Unit tests are, like mentioned in chapter 1, a form of automated testing. Unit tests are performed with the use of tools (or frameworks), for instance JUnit[3] which is a unit testing framework for Java. Like other testing methods, the purpose of unit testing is to improve the quality of the software and increase the correctness of the software [27].

Test Fixtures

A test fixture in a unit testing class is the set-up code needed to configure or place the system under test into the correct state before or in between the execution of tests. Test fixtures can manifest in the form of a single setup method, or be comprised of all the initialised member fields of a unit testing class common to all test methods, a so called implicit setup.

2.3.3 White-Box Testing

White box testing examines the structural qualities of a program [1]. Examples of white-box testing include examining how much of the system under test is executed by test suites. Such analysis is also known as code coverage analysis.

[3]http://junit.org

2.3.4 Code Coverage Analysis

Code coverage analysis is a white-box testing method which analyses the structural qualities of code [28]. During the execution unit tests, a code coverage analyser keeps track of various aspects of the code executed by the unit tests. The resulting instrumentation makes a number of metrics available to the developer. Examples of such metrics include:

- Instruction coverage, tracks the number of machine instructions executed by tests out of a total number of instructions
- Branch coverage, tracks the number of branches (boolean expressions such as *if* or *while* statements) evaluated to both true and false. A missed branch constitutes for instance a boolean expression only evaluated to true during test execution, but not false.
- Line coverage, tracks the number of lines of code executed by tests out of a total number of lines. This data is not always accessible due to the nature of individual code coverage analysis tools.

JaCoCo

JaCoCo is a free, open-source code coverage analysis tool for Java. It gathers branch, instruction and line coverage for each test class among other coverage metrics, and saves it to disk in a convenient CSV[4] format, which is easy to read.

2.3.5 Test Smells

Several researchers have contributed to both the definition of various test smells and detection of test smells within test code. Van Deursen et al. [5] defined 11 different test smells and how to mitigate them in 2001 by building on Martin Fowler's previous work on source code smells in 1999. Other scholars have contributed to the field with tools that aim to detect test smells within test suites. In 2013 Greiler, van Deursen and Storey [29] created an static test smell detection tool called TestHound. TestHound

[4]http://en.wikipedia.org/wiki/Comma-separated_values

extracts a number of facts from Java test source code at compile time, and categorises these facts into test smell indicators. A set of metrics defined by Greiler et al. [29] then makes use of these test smell indicators to detect various test fixture-related test smells. TestHound comes in the form of an executable application, which needs user input in order to perform the analysis. Greiler et al. [29] note that in future releases, they intend to create a Maven plugin for TestHound, but as of March 2015, no such release has occurred. TestHound and by detects the following test smells:

General Fixture

Many unit testing frameworks have the option of running a test fixture before the test is run. For instance, the JUnit framework has an annotation called `@Before` that can be used to mark a method that will be executed before a unit test is run. The smell occurs when the test fixture becomes too general. This makes the test harder to understand and can cause the test to run slowly as too much work is performed in the test fixture. This can be mitigated by stripping down the test fixture and have it only contain code used by all the tests and inlining the remaining code in the test method that uses it.

Test Maverick

The Test Maverick smell is related to the General Fixture smell described above. Test methods in test classes which contain implicit setups should utilise said implicit setups. If these test methods do not utilise any of the implicit setup fields they are considered test mavericks. The existence of test mavericks in test classes implies that there is setup code being executed without it being needed. Test mavericks also reduce the understandability of the test class. Since test mavericks do not depend on the implicit setup, they can be safely extracted into another class.

Dead Field

Dead fields in a test class are fields that are not used by any method or field in the class. Dead fields can reduce code legibility as well as adding clutter and possible side effects if initialized. Refactoring dead fields is simple; one simply removes the dead fields.

Obscure Inline Setup

Unit tests can serve as documentation for the system under test. Inline setup code that not only contains the steps required to run and understand a test, but also code irrelevant to the steps themselves runs counter to the documentation benefit of unit tests. Such unit tests are referred to as having obscure inline setups. In order to fix obscure inline setups, one can move irrelevant setup code into a separate method, or move setup code common to more than one test into an implicit setup belonging to the test class.

For more detailed overview of test smells the reader is instructed look up van Deursen's work on the topic [5]. Bavota et al. published a study in 2012 which empirically shows that test smells are widely spread and that most of the test smells affect the the maintainability of test suites and production code negatively [30].

3

Research Methodology

This thesis uses design science as a research method and is based upon the regulative cycle framework created by Wieringa [31]. The regulative cycle provides detailed instructions on how to execute the design science research methodology.

3.1 Design Science Research

Design science research is an iterative problem solving method. Its core concept is the creation of an artifact (a model, prototype or an implementation) where deeper understanding and knowledge of the practical problem is gathered while building the actual artifact itself and during the application of the artifact. G-Unit, the tool created in this study, is a test analyzing tool with a gamification layer on top. The artifact is created to address a certain practical problem, and in order to acquire knowledge from the artifact, it must be analyzed and evaluated. It is quite important that the artifact is innovative since it has to solve a previously unsolved problem or solve a previously solved problem in a better way. The process and the artifact construct a problem space where researchers can apply their methods and mechanisms in order to search for a solution. Researchers conducting design research must be able to present their results to both a technical (programmers, other researchers) and non-technical audience (management) [32].

3.1.1 Context

The artifact in this thesis is G-Unit, the test analyzing tool. Advania is currently facing problems with motivating unit test writing and faces uncertainty of test quality. The artifact is intended to address these problems by combining gamification elements with test evaluation methods, and the tool will be evaluated in conjunction with the developers working at Advania. After evaluation and analysis of the artifact the results will be presented in the form of this study.

Practical problems and knowledge problems respectively are defined by Wieringa [31] as: “I define a practical problem as a difference between the way the world is experienced by stakeholders and the way they would like it to be, and a knowledge problem as a difference between current knowledge of stakeholders about the world and what they would like to know“. The problems Advania is currently facing in terms of testing are practical since Advania wants to change the way testing is conducted within the organization. Although a knowledge problem might be nested within the practical problem, in the form of lacking testing knowledge by the developers, it does not change the nature of the main task. Since the purpose of design science research is to solve practical problems [31] and Advania’s testing problems are indeed practical on the top level, this research methodology is a good fit for this thesis. The solution presented in this thesis could also be considered quite innovative as substantial work hasn’t been conducted on the gamification of unit testing.

3.1.2 Design Science Research vs. Action Research

During the first decades of the 20th century, researchers have been looking into the similarities of design science research and action research [33]. Some conclude that their similarities are substantial [34] while others even come to the conclusion that one can not differentiate between them [34].

The differences between the methodologies lie in their background. Design science research originates from engineering and computer science research while action research originates from social studies [35] and focuses on the researcher as an active participant in solving practical problems [34]. Design and usefulness of the created artifact is the core concept of design science research but in action research the core concept is "[...]

the focus of interest is the organizational context and the active search for problem solutions therein" [34]. Design design science research was chosen as the methodology for this study because of the emphasis put on developing an innovative artifact designed to combat the motivational problems discussed in Section 1.3. Also, as this study focuses more on the created software artifact and its viability to solve the practical problems rather than focusing on the researchers themselves, design science research is a better fit.

3.2 The Regulative Cycle Framework

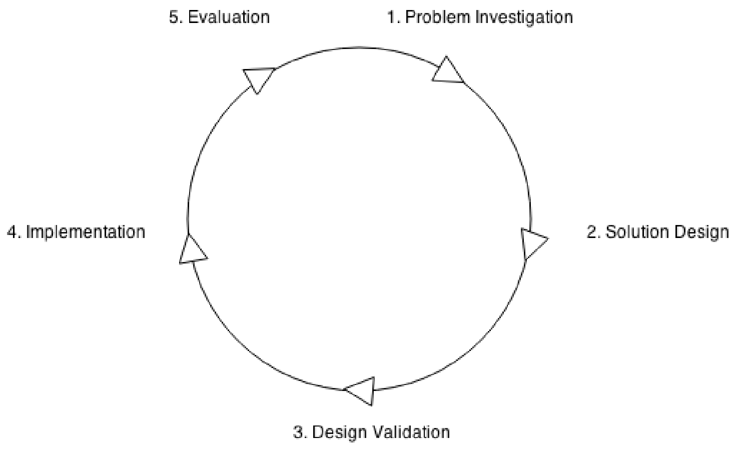


Figure 3.1: The Regulative Cycle

The applied framework in this thesis is the regulative cycle, provided by Wierenga[31]. By Wierenga's definition of design science research and the regulative cycle, each design science research project should be structured as a set of problems where the top level problem is always a practical problem and the regulative cycle framework provides a logical structure to solve these problems. The regulative cycle consists of four phases (or steps), the first one being investigation of a practical problem. The second phase is solution design, followed by a design validation and finally implementation of the solution. An optional phase is the evaluation phase, where the implementation is evaluated and can serve as a base for the first step in a new regulative cycle. This thesis will incorporate the evaluation phase for three iterations. The evaluation phase of iteration one will be used as a base for the regulative cycle in iteration two, the evaluation phase of iteration two will be used as a base for the regulative cycle in iteration three. The evaluation phase in iteration three will be used to evaluate the iteration, but will not serve as a base for a new cycle because there are only 3 iterations in this study. The last evaluation will serve as the strongest evaluation of G-Unit as the tool will have all of its test analyzing and gamification elements implemented in iteration two, allowing the polishing of features in iteration three.

3.2.1 Problem Investigation

In a problem investigation, researchers seek to gather knowledge about a problem without actually acting on the problem. The goals of the investigation are to diagnose the problem by describing and explaining it and possibly to predict the outcome if nothing is done to address the problem identified. There are four different kinds of problem investigations which differ in terms of what drives the investigations. First, an investigation can be "problem-driven" when stakeholders aren't certain about what constitutes the problem and an analysis of the problem is required in order to solve it. Second, an investigation can be "goal-driven". This is when no real problems are experienced per se, but for some reasons a change is needed. Third, an investigation can be "solution-driven". This is the case when investigators are in search of problems that a new technology could potentially solve. Finally, an investigation can be "impact-driven". This is when researchers focus on evaluating the outcome of previous methods or actions instead of focusing on a new design or a solution [31].

The problem investigation conducted in this study was split up in to four different parts.

- Exploratory work in the form of a literature review was conducted in order to identify common problems that organizations are facing in terms of testing, especially unit-testing. This part of the problem investigation could be categorized as "impact-driven", since an attempt was made to identify and evaluate problems that the software industry has been facing in the past years in terms of testing and the countermeasures taken by the industry in order to fight these problems.
- Identifying how these problems could actually be addressed through a software artifact. The second part could be classified as "solution-driven" since the authors

were researching which kinds of problems could be solved with a software artifact, although it had not been created at the time.

- Identifying the unit testing problems Advania was facing in collaboration with a project manager at Advania, both from the social perspective and technical perspective. Identification of Advania's testing problems was "problem-driven" since interviews were needed to pinpoint exactly which kinds of problems Advania was facing in terms of testing. After a series of meetings and discussions through email, the following problem areas were identified within Advania's testing infrastructure:
  - Very few tests are being written
  - Advania is unsure of the quality of the tests which are actually written
  - Motivation to write tests is lacking

  These problems are common within the industry as discussed in chapters 1 and 2.
- Identification of problems within the G-Unit tool itself and is the only investigation that is continuous throughout the project, serving as the last step in the first problem investigation phase and the first step in the following iterations. Developers were sent questionnaires about the usability and the effectiveness of the tool in order to identify problems lying within it. The improvement of the tool itself, was classified as "problem-driven", since the developers help was needed in order to reveal the actual problems with the tool.

3.2.2 Solution Design

In the context of Wierenga's framework, a design is a plan where "[...] means to an end is laid down." [31, pg. 4]. At the end of each solution design path, stakeholder goals should be met and how to reach these goals is the proposed solution. The proposed solution has to be communicated to all stakeholders and therefore has to be specified in some way, be it natural language, diagrams or by other means. The solution design is classified as a practical problem, rather than a knowledge one, since it describes the stakeholder's approach on how they intend to change the world.

In this study, the solution design is based upon the earlier problem investigations that had been conducted and the authors' ideas on how to address these problems through a software artifact. Feedback from Advania's employees was important in this step, although their feedback on the actual artifact is only available during an iteration and after it has taken place, since the artifact was not presented to them until the start of iteration 1.

Although some requirements were laid down in the problem investigation phase, some further work was conducted on those specifications during the solution design phase. In this phase further documentation was added in the form of diagrams. These diagrams were created in order for the authors to understand each others' implementation ideas regarding the artifact and to gain a more coherent picture of the system flow. Before a solution could be designed for an iteration, the requirements had to prioritized into each of the iterations since pressure to meet deadlines would not allow for all requirements to be implemented in the first iteration.

Advania was not interested in heavy documentation regarding the artifact (excluding usage-documentation) and therefore the requirements specification of the artifact was not sent to Advania. Instead, the artifact and its features were presented to the developers before it was put to use.

Wierenga concludes that solutions are often not fully specified before they are actually validated and implemented. Hence the product of this phase is not necessarily a fully specified solution, but rather a design that will eventually be completely specified and implemented at the end of the regulative cycle.

3.2.3 Design Validation

The design validation phase is needed in order to validate that the prior phase has indeed resulted in a design that will bring stakeholder closer to their goals. Three knowledge questions should be considered in this phase [31]:

- Internal validity: "Would this design, implemented in this problem context, satisfy the criteria identified in the problem investigation?"
- Trade-offs: "How would slightly different designs, implemented in this context, satisfy the criteria?“
- External validity (a.k.a. sensitivity analysis): "Would this design, implemented in

slightly different contexts, also satisfy the criteria?“

These questions are knowledge questions and their answers are propositions that the validators (in this case, the authors) claim to be true. These are predictions set forth by the validators in their attempts to predict the results of their solutions in a certain context [31]. Along with these three questions, each iteration has its own specific additional questions.

After the design of the software artifact these questions were kept in mind in order to visualize the results based on the artifact design. The actual artifact itself never underwent validation by Advania's employees and the validation was only performed by the authors. The answers for each of the iterations can be found in Sections 5.1.3, 5.2.3 and 5.3.3, for iterations 1, 2 and 3 respectively.

3.2.4 Implementation

In the implementation phase, designed and validated solutions are implemented. However, the implementation may vary both in terms of details and complexity. A software problem can for example have many solutions which may be expressed in different ways. For instance, two different designers designing a solution for the same problem can come up with different implementations, one designing a paper prototype while the other implements a working software solution.

The implementation consists of a software solution for all the iterations. However, their complexity level varies, as more features are added in each iteration. Some components are also removed while others are added, based on the earlier steps in the regulative cycle described in this chapter.

3.2.5 Evaluation

Although the evaluation phase is not a formal phase in Wierenga's framework, it was decided to incorporate the phase in the study for reasons discussed in section 3.2. The phase provides an opportunity to evaluate previous work and, as Wierenga suggests, the phase can serve as a base for a new regulative cycle performed in a new iteration.

The artifact was evaluated through questionnaires for the first (Appendices B.1, B.2 and B.3) and second iteration (Appendices B.4 and B.5). The questionnaires focused on gathering feedback from the developers that used G-Unit in their work environment. The third iteration was evaluated by interviewing the developers.

The results from the evaluation phase of each of the iterations can be found in Chapter 6.

3.3 Developers, Interviews & Surveys

This study used both interviews and surveys to collect qualitative data. The surveys are of two different kinds, both in form of a daily questionnaire that the developers completed each day during an iteration and then a post-iteration survey. The surveys were designed based on the guidelines provided by Leung [36]. The survey data can be found in Appendix B.

The purpose of interviews is to collect qualitative data that can not be acquired by quantitative means [37]. This study is largely of qualitative nature and therefore an interview was conducted in order to provide insight in to the developers opinions, thoughts and feelings [37]. Interviews can both be of individual nature, where a single subject is interviewed about a certain topic, or group interviews where several subjects discuss topics introduced by the interviewers [37]. The interview used in this study was a group interview with two subjects and two interviewers. The interview was a video call through Skype, in which the interviewers recorded the audio. The interview was 20 minutes long. Interviews can have different structures, such as fully structured, semi-structured or unstructured [38]. In a fully structured interview the questions are all predetermined and their order is preordered as well. During an unstructured interview the interview questions are in form of a general concern and/or interest from the researcher [38]. Semi structured interviews have some pre-planned questions but they are not necessarily asked in the predetermined order and the development of the interview can dictate the order of the questions. Semi structured interview also allow for improvisation and allow the interviewers to examine the studied objectives further on the fly [38]. The group interview conducted in this study was semi-structured. The interview questions can be found in Appendix B. The participants in both the interview as well as the surveys were all software programmers who have between 1-3 years of work experience.

The study had a varying amount of participating developers. During the first iteration, there were three developers who answered a survey before the start of the iteration. They also answered a daily questionnaire during the iteration. The iteration was 4 days long and thus resulted in 12 answers to the daily survey. The data is in Appendix B. They also answered a post-iteration survey conducted after the iteration was finished. The data from that survey is not in the appendix, but is displayed in Table 6.1. During the second iteration, there was only one participating developer which answered a daily survey. The iteration was four days, resulting in four answers to the daily survey. The last iteration had two participating developers. There were no surveys in this iteration, but instead, a semi structured interview was conducted like discussed in the paragraph above.

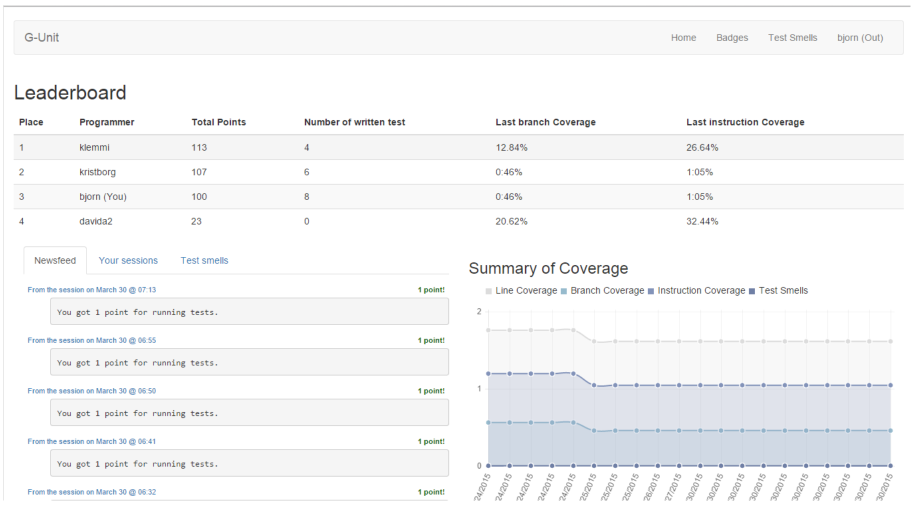
4

G-Unit

This chapter presents the G-Unit gamification tool and its underlying concepts.

4.1 Overview

G-Unit is a gamification tool for unit testing in software development. It consists of two main components. Firstly, the G-Unit Maven Plugin collects data from a single test session, which consists of results from JUnit test executions, code coverage data from the JaCoCo coverage tool, and test smell analysis data from TestHound. Secondly, the G-Unit Service receives the results from the Maven plugin. These results are fed into a gamification rule engine. The rule engine then calculates statistics from each individual test session, and awards points to the respective developer in accordance to a set of gamification rules listed in Section A.3. The G-Unit Service also serves a web site which provides gamification feedback to the developers. A number of gamification elements were utilized. A leaderboard is provided to the developers, where they are ranked according to the total amount of points accumulated. A graphical overview of their achieved code coverage and test smells enables the developers to see how their progress has been over time. The developers are able to view each of their past test sessions, and compare them to their current standings. Each developer is provided a personalised news feed, where a summary of the the points awarded is presented by the gamification engine. When certain conditions are met by a developer, the gamification engine awards that developer a badge according to a few rules described in section A.3. Some badges can be earned many times. A developer's accumulated badges can be viewed on their individual user profiles.



Leaderboard

Place	Programmer	Total Points	Number of written test	Last branch Coverage	Last instruction Coverage
1	klemmi	113	4	12.84%	26.64%
2	kristborg	107	6	0:46%	1:05%
3	bjorn (You)	100	8	0:46%	1:05%
4	davida2	23	0	20.62%	32.44%

From the session on March 30 @ 07:13 1 point!

```
You got 1 point for running tests.
```

From the session on March 30 @ 06:55 1 point!

```
You got 1 point for running tests.
```

From the session on March 30 @ 06:50 1 point!

```
You got 1 point for running tests.
```

From the session on March 30 @ 06:41 1 point!

```
You got 1 point for running tests.
```

From the session on March 30 @ 06:32 1 point!

Figure 4.1: G-Unit front page - News feed

Advania uses IntelliJ IDEA[1] as their primary IDE. An IntelliJ plugin is provided whose main goal is providing instant feedback of the gamification results to the developer.

4.1.1 Flow

In general, the use of the system could be described with the following general workflow:

1. The developer writes a unit test, which can be executed with JUnit
2. The developer executes the Maven goal "mvn verify", which runs the JUnit test runner, JaCoCo code coverage analyzer, and lastly, the GUnit maven plugin.
3. The GUnit maven plugin posts the accumulated test data to the backend
4. The data is analyzed by the backend, rules executed and badges awarded accordingly.

[1]https://www.jetbrains.com/idea/

5. The user refreshes the updated score page in order to view their position on the leaderboard, and to see any new badges.

4.2 Implementation

G-Unit is mainly implemented in Java 1.8 and uses a MySQL database. The G-Unit Service is using a REST[2] focused web-application framework called Dropwizard[3] as a basis. Parts of G-Unit require earlier versions of Java, both 1.7 and 1.6. The IntelliJ plugin is written against Java 1.6 because of the plugin API requirements. All Maven-related code is written using Java 1.7 for compatibility reasons related to Advania. The client-side web application itself is built with the AngularJS[4] web application framework and uses the Twitter Bootstrap UI[5][6] frameworks for various user interface components.

As the TestHound application has no integration with Maven, a Maven plugin was written which invokes TestHound with the required information, and serializes the result to an XML file, which the G-Unit Maven Plugin then reads.

Maven is used to build and package G-Unit. All the maven projects are published to a Maven repository created specially for this thesis, where the developers at Advania can easily access and update their versions of them. This greatly simplifies the deployment of new plugin versions in relation to each iteration, as the developers can change the version they are using to the newest version, and Maven fetches it automatically.

[2]http://en.wikipedia.org/wiki/Representational_state_transfer
[3]http://dropwizard.io/
[4]http://angularjs.org
[5]http://getbootstrap.com,
[6]http://angular-ui.github.io/bootstrap/

4.2.1 Design

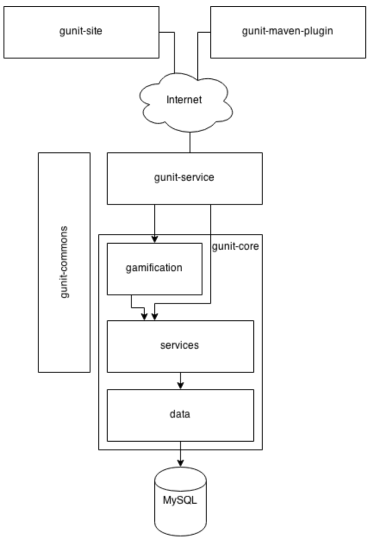


Figure 4.2: System architecture

G-Unit is comprised of 9 modules:

1. **gunit-maven-plugin**: The G-Unit Maven Plugin component, which accumulates and sends data to G-Unit Service.
2. **gunit-parent**: Parent project which decides the order of compilation.
3. **gunit-core**: The core gamification code and database interaction code.
4. **gunit-service**: The G-Unit Service component, whose main functionality is to receive test session data from G-Unit Maven Plugin.
5. **gunit-site**: A client-side web application which provides feedback to the developers, bundled wih G-Unit Service.
6. **gunit-service-client**: A client utility for interacting with G-Unit Service.
7. **gunit-commons**: A library which contains code common to several other modules.
8. **testhound-maven-plugin**: A maven plugin which executes Michaela Greiler's TestHound test smell detecting tool.
9. **intelli-gunit**: An IntelliJ IDEA plugin which notifies a user of any newly processed data by the gamification server.

G-Unit uses a client-server software architecture, with the G-Unit Maven Plugin representing the client in that context. The G-Unit Service uses a layered design illustrated in Figure 4.2.

4.2.2 User Interface

The user interface of the application consists of six main parts:

- Front page (Figure A.1) containing a leaderboard, test session listings, test smells overview and code coverage visualization displayed with a graph.
- User profile overview (Figure A.4)
- Test session overview (Figure A.5)
- Badges overview (Figure A.6)
- Test smells overview (Figure A.7)
- IntelliJ notification plugin

Since the tool was deployed on site at an Icelandic department within Advania, the user interface was in Icelandic during the iterations. For clarification purposes the user interface in the screen shots has been translated from Icelandic to English using a combination of Google Translate[7] and manual translations.

On the front page (Figure A.1) a user can view a leaderboard which contains his name, position within the leaderboard, total points along with last sessions' branch and instruction coverage. The front page also contains a list of the users test sessions and information regarding that test session (Figure A.2). The user can also get a per-test class view of her test smells, and a description of each test smell (Figure A.3), by clicking any of the detected test smells. The graph presented on the front page visualizes the user's code coverage in terms of branch, line and instruction coverage.

The user can navigate to a user profile overview (Figure A.4) by clicking on the user name. The user profile pages can be viewed by any user. Each profile page displays a user's earned points, badges, number of tests written, as well as statistics regarding their last branch and instruction coverage.

A user can navigate to the test session page (Figure A.5) by clicking on a test session on the front page. The test session page displays information regarding the selected test session in terms of code coverage and points acquired from that session.

[7]http://translate.google.com

The badge page (Figure A.6) displays a number of badges available for the user to claim. Each badge has an associated task that the user has to solve in order to claim the badge. The badge page does not display all the available badges in the system, as there is a possibility for a badge to be hidden, meaning that the only way for a user to know about a hidden badge is to solve its associated task and thereby claiming it.

The test smells overview (Figure A.7) lists the four different test smells detected by TestHound and used by G-Unit to score the users. It was added in the third iteration as a means of better clarifying what test smells are, even though that information was previously available in the second iteration, as a part of the test smells listing on the front page. A user could click on each detected test smell to get a description of the test smell and how to refactor it. This information is available regardless of whether the users have introduced the test smell in their test suite or not.

The IntelliJ IDEA plugin displays a notification inside the IDE whenever anyone submits a test session to the server. The notification is in the form of a message bubble displayed on the bottom of the IDE, and shows who submitted the test session, how many points they had gotten, and a link to the G-Unit page. From a gamification perspective, the plugin serves as a provider of instant feedback to the developers, congratulating them when they themselves receive points for their test sessions, and letting them know when their competitors receive points.

5

Iterations

The following subsections present the work done during the iterations in the context of the regulative cycle.

5.1 Iteration 1: Code Coverage

The purpose of iteration 1 was to create a working software artifact (the G-Unit tool) that would cover most of the requirements gathered in the first problem investigation phase. This version of the artifact would serve as a basis for future versions of the artifact. The architecture of the artifact was designed to be easily upgradeable since more features would need to be added to the artifact in future iterations. The GUI was designed so it would not change much in terms of structure between iterations, for example the location of buttons, graphs and tables. The artifact produced in the first iteration included a gamification engine which based its calculations on code coverage metrics and JUnit test result data. As such, the focus was on the external qualities of unit tests, namely their coverage of the system under test. It was also quite important to create a working software solution, even though it only had a part of the requirements implemented, in order to demonstrate the implementation ideas to Advania. An agreement with the company had been made to carry out three iterations and therefore having a working software artifact in iteration 1 would be optimal. Getting feedback on the features along with the GUI in iterations one and two was important. The third iteration could then be primarily dedicated to polishing and bug fixing.

5.1.1 Problem Investigation

This phase was the first step in the study. When the authors were formulating the topic for the study they were interested in testing, but were unsure which testing problems would be the most interesting to them. Since unit testing was the only field of testing the authors both had some prior experience, aside from manual testing, they decided to focus on organisational problems in the context of unit testing which the authors both found interesting. A literature review was conducted on unit testing, both in organisational context and from a technical aspect. After the initial literature review, the following problems were identified in the context of unit testing within organizations:

- Testing is considered a boring task within organisations [14] [23],
- Test source code is just as susceptible to design flaws as system source code [30]
- Test smells are widely spread and that most of the test smells affect the the maintainability of test suites and production code negatively [30]

The software industry is without a doubt facing other challenges in terms of testing but these problems are the most interesting.

In parallel to the literature review the authors were in contact with management at Advania. A brief informal interview, along with email communication with a project manager at Advania revealed that Advania was facing the above problems like many other software organizations. In the case of the educational software development department within Advania, where the tool would eventually be deployed, the developers had written almost no unit tests for their product, their experience in unit testing was little and their motivation to write tests was low.

Since the authors brought this problem to Advania's attention in the first place, Advania did not have a solution to their problems in mind. From a certain viewpoint that was beneficial to the authors' thesis work as Advania gave them complete creative freedom to design a solution to Advania's problems. This also aligned with the authors' ideas of working within design research, as the software artifact created in information system design research has to be innovative [32].

5.1.2 Solution Design

Because of the creative freedom provided by Advania it was decided to design a gamification based solution to address Advania's testing problems. As discussed in Chapter 2.1.3, some ethical considerations must be considered and as discussed in that section, a gamified working environment could possibly create friction between coworkers.

The solution was to create a gamification rule system that would attempt to encourage developers to write more tests with a higher quality. The originial intent was to incorporate both code coverage and test smell detection in Iteration 1 but due to time constraints the incorporation of test smell detection was not an option. Code coverage on its own is not the ideal indicator for quality of a unit test, but keeping in mind that Malayia et al. argue that thoroughly covered modules are less likely to contain errors [11], it was a good enough indicator for Iteration 1. The gamification system designed in this phase rewarded developers for:

1. Writing at least one unit test per day by rewarding the developers in the form of a large point boost and a badge
2. Writing a certain amount of tests during an iteration by rewarding badges
3. Reaching a certain level of branch and instruction coverage
4. Running unit tests
5. Point deduction for lowering code coverage and abuse of concept nr. 1

The full set of rules and how they are precisely defined can be found in Appendix A.3.1.

Designing a balanced gamification system proved to be quite a challenge. The issues revolved around balancing the points for each gamification element and problems related to code coverage rewards are discussed in the evaluation section.

5.1.3 Design Validation

The time spent on this phase was minimal compared to the other phases. The validation was conducted by the authors as Advania was more interested in observing the actual tool in action instead of involving Advania in the development process. During this phase Wieringa's validation questions were considered among others.

Question 1 (Internal Validity): Would this design, implemented in this problem context, satisfy the criteria identified in the problem investigation?

The features implemented in Iteration 1 would only partially satisfy the criteria identified in the problem investigation phase as the system did not include any test smell detection. However, the features that were implemented were enough to create a gamification system that would have the potential to influence the behaviour of the developers to write more tests with high code coverage.

The phrasing of this question is broad. During the design and validation phase other more specific questions were considered. Those questions can be classified as sub-question of this question. The main two sub-questions that were considered were

Sub-Question 1.1: Would this design address Advania's testing problems?

Yes, by increasing the developers' engagement in testing through gamification.

Sub-Question 1.2: Would this design encourage developers to write more and/or better tests?

Yes, by rewarding the developers handsomely for submitting at least one test a day while also providing them with statistical information regarding their unit tests.

Question 2 (Trade-offs): How would slightly different designs, implemented in this context, satisfy the criteria?

Incorporation of test smell detection would have satisfied the criteria by enabling the detection of test smells and thus satisfying the requirements gathered in the problem investigation phase. However, due to time constraints test smell detection was not an option in iteration 1. The main focus of this iteration was to have a working software artifact and the creation of gamification rules on top of code coverage statistics was both technically easier and less time consuming than test smell detection. Therefore it was decided to cut test smell detection from iteration 1 in order to be able to deliver the artifact on the predetermined release date.

Question 3 (External Validity): Would this design, implemented in slightly different contexts, also satisfy the criteria?

The gamification system is highly coupled with unit testing so switching from unit testing to another type of software testing would probably not work. However this system could for example be incorporated in a different environment, for example in an educational context and achieve similar results.

5.1.4 Implementation and Calculations

The analysis of unit tests is split into two parts in this iteration: code coverage calculations and gamification calculations. The code coverage part analyses unit tests with the JaCoCo code coverage analysis tool. The tool collects, among other things, line coverage, branch coverage and instruction coverage, which is collected by the G-Unit Maven Plugin and sent to the gamification service. Although the tool reports line, branch and instruction coverage, the gamification service only uses branch and instruction coverage for gamification calculations. A listing of the rules used to calculate a user's score per test session can be found in appendix A.3.1.

Further technical implementation details are discussed in section 4.2. All modules discussed in that section were completed in the first iteration with the exception of the TestHound module which was introduced in iteration 2.

In this iteration one of the thesis authors was on location in order to set up G-Unit on the developers computers. Technical support was provided during the iteration and the developers were taught how to use the system. The authors were not able to be present on location during iteration 2 and 3 due to high cost of travel between Iceland and Sweden. However, the user experience did not change much from iteration 1 to the subsequent iterations and therefore a new technical walkthrough between iterations was not needed. The tool can also be updated through the internet which eliminated the need for on-location set up for iteration 2 and 3.

5.1.5 Changes from Design

During this phase it became apparent that the gamification system that had been designed in the design phase was unbalanced and needed adjustments. The gamification rules were adjusted on the fly during this phase since it was discovered that the rules did not make as much sense as they did during the design phase. However, as discussed in the evaluation section below, the system still suffered from balancing flaws after these adjustments.

5.1.6 Evaluation

The evaluation of this iteration was conducted by on-location observation and questionnaires. The iteration lasted for four workdays. Each developer (three in total) answered a daily questionnaire after each workday in the iteration, resulting in 12 answers to every question in the questionnaire. Another survey was also conducted after the iteration had finished to gather additional feedback. The questionnaires were designed with the guidelines provided by Leung [36].

The results from the evaluation of iteration 1 can be found in Section 6.1.

5.2 Iteration 2: Test Smells

This iteration focused on the addition of test smell detection through the use of TestHound and general improvements on the G-Unit tool based on the feedback gathered in iteration 1. After the addition of test smells all the desired features were present in the tool. The length of this iteration was 4 workdays just like iteration 1.

Unfortunately this iteration was plagued with issues regarding developer participation. One out of the three developers was able to participate. One developer was on vacation during the iteration and another developer was unable to participate in the iteration for organizational reasons. The evaluation of the iteration would therefore be based solely on one developer. Since the developer had no one to compare to during this iteration the evaluation gamification part of the application was not as strong as it could have been if there had been more participants. This also applies to evaluation of other parts of the system. The developer was asked to evaluate the changes made to the gamification system although he was the only one participating. The feedback regarding the leaderboard particularly suffered since comparing one's points to inactive participants has little meaning. Therefore the evaluation of the gamification part was focused on the improved point and badge system.

5.2.1 Problem Investigation

During this phase the feedback from iteration 1 was analyzed. Like discussed in 6.1 the developers wanted more instant feedback and a clearer point system. Problems related to the integration of TestHound were the other significant part of this problem investigation. The main problems with TestHound's integration was that the source code wasn't available which sparked implementation issues.

5.2.2 Solution Design

The rules of the gamification part of the tool were redesigned based on the feedback gathered from iteration 1 and can be found in the appendix. In addition to the old rules, new rules based on test smell detection were created. This would allow the gamification system to take both code coverage and test smell analytics as input. The badge system was also redesigned in order to provide better information in regards to how developers could achieve badges within the system.

5.2.3 Design Validation

Again, Wieringa's questions were consider along with other questions designed by the authors.

Question 1 (Internal Validity): Would this design, implemented in this problem context, satisfy the criteria identified in the problem investigation?

Yes, because all of the planned requirements were present this iteration's design.

Sub-question 1.1: Does the new score system reflect the developers feedback?

Yes, more instant feedback was implemented along with modifications on the score system based on the developers feedback.

Sub-question 1.2: Does the introduction of TestHound's smell detection help the developers maintain high test quality?

Yes, by pointing out faults within the test code and rewarding developers for removing test smells from a test class.

Question 2 (Trade-offs): How would slightly different designs, implemented in this context, satisfy the criteria?

More gamification elements related to the test smells could potentially influence the developers behaviour in a positive way. However, certain trade offs had to be made in this study due to time constraints, so more test smell related gamification rules were not created. It was estimated that the test smell rule introduced in iteration 2 would be enough motivation for the developers to get rid of their test smells.

Question 3 (External Validity): Would this design, implemented in slightly different contexts, also satisfy the criteria?

As before, the system is highly coupled to unit testing. If the context was still unit testing but in a different environment, this design could satisfy the criteria.

5.2.4 Implementation

During this phase the evaluation of the unit tests is split into three parts: Code coverage analysis, test smell detection and finally gamification calculations based on the gamification rules defined in the design phase (which now included test smell based rules). As before, the code coverage part is handled by JaCoCo and the gamifcation rule engine calculates the points awarded for each test. The result from TestHound's analysis is presented in the UI where each developer has an overview of his test classes and whether the test classes contain any test smells. If a class contains smells, the tool lists the smell and details related to that smell. A badge page was also introduced in this iteration, where developers can view available badges and how to achieve them. As before, more implementation details and a typical workflow description can be found in Chapter 4.

5.2.5 Evaluation

As discussed in Section 5.2 this iteration was only evaluated by one developer. The evaluation was done through both daily questionnaires and a post-iteration questionnaire. The results from the evaluation can be found in Section 6.2.

5.3 Iteration 3: Instant Feedback

Iteration 3 added instant feedback in the form of an IntelliJ plugin, which notified the developers whenever any of the developers got points for their test respective test sessions. Alongside the addition of instant feedback, iteration 3 added various small bug fixes and polishing. As with the previous iterations, the length of iteration 3 was 4 days.

5.3.1 Problem Investigation

Problems with the instant feedback were identified along with general bug fixing within the tool's source code. Some work had been done in the first implementation phase regarding feedback within the IntelliJ editor, but had remained unfinished until iteration 3. The feedback is displayed within the IntelliJ editor through the Intelli-Gunit module, discussed in sections 4.2 and 4.2.1. Some work had been performed on this module in the first iteration. The idea in the first iteration was to display the leaderboard within the IntelliJ editor as well as on the website, but that idea was abandoned for various reasons. One reason being that IntelliJ's plugin architecture is both old and documentation was scarce. Another reason was that although the authors were able to construct a leaderboard table within IntelliJ, it was extremely unstable and was estimated that it could possibly interfere with the developers work in terms of crashes and other unstable behaviour. However, the code needed to create notifications within IntelliJ was found on GitHub after an extensive search through open source IntelliJ plugin projects that are available on the site. That code was more reliable than the old code and therefore was deemed fit for the tool. These notifications would enable the system to notify developers within the IntelliJ editor when new relevant information was available regarding the test code base. Information regarding test smells within the G-Unit tool was also deemed insufficient.

5.3.2 Solution Design

The new notification system was designed to notify developers when new information was available within G-Unit. The system notifies developers when they send new test data to the server or when other developers submit new test data. A new section on the tool's website was designed in order to ease access to test smell information. A new section within G-Unit's website was also designed so developers could look up both descriptions and mitigation tactics regarding the test smells.

5.3.3 Design Validation

As in the previous iterations, Wierenga's validation questions were considered along with other questions created by the authors. Wierenga's third questions, regarding external validity, was discarded this time since the answer is the same as in iteration 2.

Question 1 (Internal Validity): Would this design, implemented in this problem context, satisfy the criteria identified in the problem investigation?

Yes, as the new notification system addresses the feedback issues discussed in sections 5.3.1 and 6.2.

Sub-Question 1.1: Does this design provide the developers with enough information to eliminate test smells?

The new test smell section of the tool's website provides both information regarding description of test smells as well as how to eliminate them. This should provide the developers with enough information to elminate the test smells.

Questions 2 (Trade-Offs): How would slightly different designs, implemented in this context, satisfy the criteria?

Having a leaderboard along with the notifications within the IntelliJ editor would have satisfied the criteria, if and only if the leaderboard would be reliable enough. However, since reliability and consistency were valued more than the addition of this single feature, the leaderboard was scrapped.

5.3.4 Implementation

The architecture of the system did not change much between iteration 2 and 3 and the main functionality of the system was not changed at all. The code needed for the notification functionality was partly taken from GitHub like mentioned in section 5.3. As discussed in the design section, the notifications pop up when a developer submits new test data to the server. The notifications include the name of the developer along with the total number of points awarded for the new test data. The new test smell information section on the tool's website contains an Icelandic translation of section 2.3.5. This iteration also focused on various bug fixes and some polishing of the implemented features.

5.3.5 Evaluation

Unlike the first two iterations, this iteration was evaluated through interviews instead of questionnaires. The interview guide can be found in appendix A.2. One of the developers was on vacation during this iteration and therefore this iteration was evaluated by two developers. The results from the evaluation can be found in section 6.3.

6

Results

6.1 Evaluation of Iteration 1

According to the daily survey data the tool was easy to use. In 11 instances out of 12 the developers felt that the tool was either easy or very easy to use. There was only one instance, during the first day of the iteration, that one developer noted that the tool was neither hard nor easy to use. The usability of the tool is of high importance and the fact that the developers consider the tool easy to use is positive. The tool would not be able to serve its purpose if the developers were unsure how to use the tool or were unable to use the tool at all.

The surveys indicate that the point system used in the tool motivated the developers to write more unit tests. In 10 out of 12 answers the developers noted that the tool motivated them to write more tests. This is interesting because the metrics used in the code coverage point system in Iteration 1 were a little flawed. Instead of calculating per class code coverage the implementation calculated the whole code coverage of the system, resulting in the code coverage being significantly lower than anticipated. For instance, most tests had around 0-1 percent code coverage while the rules of the system anticipated coverage to be around 10-60 percent. The result of this was that the developers got very few points for the code coverage aspect. However the system allocated points for more than code coverage and it seems that those rules were enough to motivate the developers to write more tests.

Although the developers felt more motivated to write more tests during the iteration, the results are not as positive regarding the quality aspect of the tests. Only on 6 out of 12 occasions the developers felt that the system motivated them to write better quality tests. This is most likely the result of no test smell detection being present in this iteration and the flaws contained in the code coverage metrics.

The leaderboard did not seem to cause much friction or ethical problems during this iteration. On 11 out of 12 occasions the developers did not feel uncomfortable having their points up on the leaderboard and compared to the score of other developers. A possible reason for this is that the developers were scoring roughly the same amount of points during the iteration (within 10-15 percent of each other). On one occasion did one developer feel uncomfortable about the leaderboard.

The results from the feedback survey conducted after the iteration finished are presented in the following table.

Index	Findings
5.1	Would be fun to be able to see each developers branch/instruction/line coverage in the leaderboard
5.2	Developers prefer exact points instead of grouping in order to see exact position within the leaderboard
5.3	The score system was unclear
5.4	Instructions on how to write better tests in order to score more points

Table 6.1: Findings from iteration 1

Purpose of the first two questions was to get feedback on the leaderboard as there were anticipated ethical issues with that element. Some video games [1] use grouping (groups players together based on experience point interval) instead of displaying the exact points in the leaderboard and the authors were interested if the developers would feel more comfortable using a grouping based leaderboard instead of a leaderboard that displayed the points explicitly. Based on the feedback data the developers did not feel

[1]http://blog.counter-strike.net/index.php/2012/10/5565/

that a grouping based system would be better as they would like to see their exact position. The developers also wanted to see more information in the leaderboard in terms of code coverage. They also noted that the point system felt weird and were not sure how the points were exactly allocated. Lastly, they were interested in more instant feedback and instructions on how to write tests that would yield the highest number of points. Their suggestions are discussed further in the problem investigation section for iteration 2(5.2.1).

On location observation during the iteration observed that the developers would immediately start find ways to game/cheat the system. Within the first hour of deploying the system, a developer noticed a way exploit the score system. By exploiting the Test-A-Day rule (appendix A.3.1) by writing a test without any asserts or code in it, this developer was able to score 20 points every day without doing any work. This is something that had been anticipated before the iteration started and is a continuous challenge in gamification systems.

6.2 Evaluation of Iteration 2

Although this iteration's evaluation was weaker in terms of participating evaluators, the feedback was overwhelmingly positive.

On three out of four occasions the developer felt that the tool was easy to use. On one occasion the developer felt that the tool was neither hard nor easy to use. This is positive since a tool that is easy to use will not interfere much with the regular workday of the developer and thus minimising problems related to usage of the tool.

On every occasion the developer felt that the improved score system motivated him to write both more and better tests. Although the developer had no one to compete against during this iteration, the developer had used the score system in the earlier iteration. Having that frame of reference, it was estimated that the developer was able to evaluate the new score system without other participants.

The developer was also asked to evaluate if the tool (and not just the point system) helped him write better quality tests. On every occasion the developer felt the tool helped him write better quality tests. This indicated that the quality metrics (code coverage, test smells) aided the developer in his work independent from the gamification.

The post iteration questionnaire mostly focused on the test smells and the test smell features within the tool. Before the iteration started the developer was introduced to the test smell concepts through email. The developer was asked if he understood what test smells are and how they affect the code base. The developer responded that he both understood what test smells are and how they affected the code base. The developer also had the opportunity to provide feedback on how the test smell feedback was displayed within the UI, but had no comment. The developer was also asked about the improvements made on the badge system, as it had been slightly redesigned. Now the developer could look up which badges he could achieve and how to achieve them. The developer felt that the new badge page was a positive step.

As developer participation was lacking in this iteration, the authors decided to re-evaluate the tool themselves based on the same set of questions as the developers were given. They agreed with the developer's evaluation, although more instant feedback could be helpful. They was also no way for the developers to look up information regarding test smells within G-Unit, which was not optimal.

6.3 Evaluation of Iteration 3

When asked whether the tool itself helped the developers write better tests, they both agreed that the tool helped in that regard. When asked to specify which element of the tool aided them the most, they both agreed that the code coverage part of the tool was the element that influenced their tests quality the most. However, the developers would have preferred more information regarding branch and instruction coverage, as they weren't completely sure of their meaning (the tool displayed instruction, branch and line coverage). When asked whether the tool helped them write more tests, both of the developers strongly agreed that it did. When they were asked to specify why it made them write more tests, they both attributed that fact to the gamification part of the tool. When asked to clarify how the gamification part motivated them to write more tests they both attributed that fact to the leaderboard and point system or in their words, "Nobody wants to be last in the leaderboard". Although the developers felt the gamification motivated them to write higher quality tests, they didn't feel as strong in that regard as with the quantity of the tests. They attributed this to the metrics, as they sometime felt that they were unclear and needed more fine tuning.

The developers were asked if they would like to continue the use of the tool in which they responded positively. However, they pointed out that during the study the tool was not integrated until in the last stages of the current project. The developers both agreed that the tool could have served its purpose better if the study had been launched concurrently with a project, since the architecture could have been designed with unit tests in mind from the beginning. The developers felt that gamification in general was well suited for unit testing and especially noted that a gamification tool like G-Unit exposed what other developers were up to and gave them a better overview of their project. They also noted that a gamification system encouraged them to monitor what other developers were up to since they didn't want to be in the last position within the leaderboard. The subjects also felt that if management would be committed to the gamification system, for example by monitoring the leaderboard, the engagement of the developers would be higher. Although they both felt that this could raise ethical issues and potentially raise stress levels within the development group. Other ethical issues that the developers were afraid of was that people would start to cheat the system more aggressively if management would be monitoring the leaderboard, which could prove counter productive for the project as a whole.

The authors were interested to see whether the developers felt pressured by the company to write more tests as the company had agreed to participate in the study. When this subject was brought up, both developers responded that they felt no pressure at all from management to write tests. They also mentioned that no automated testing had been in place before the introduction of the tool and attributed the increased testing activity directly to the tool.

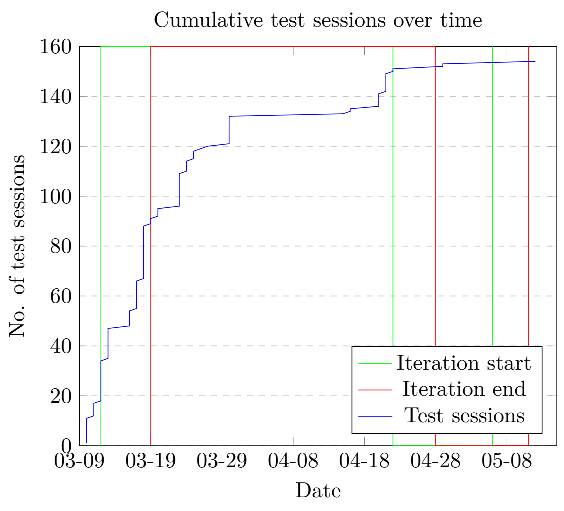
6.4 Quantitative Data

This section presents the quantitative data gathered during the study.

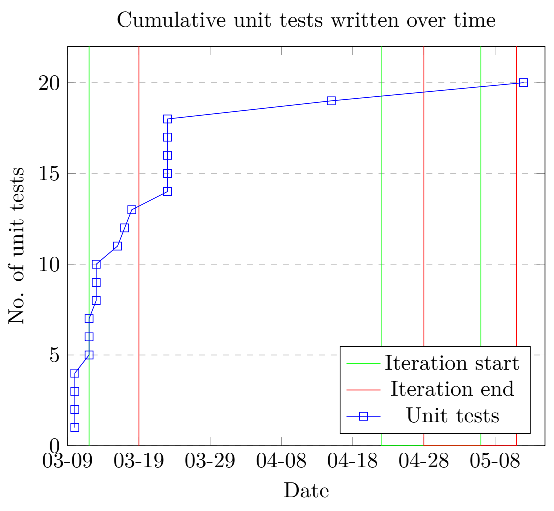
6.4.1 Number of Unit Tests Created and Test sessions

Over the course of the study there were 20 unit tests created by 3 developers. Figure 6.1b illustrates the amount of unit tests over time. Like discussed in section 7.3 and 6.3, no steps had been taken by the company to increase automated testing prior to the study and therefore no unit tests were present within the system before the study. Therefore the study seems to have succeeded in its attempt to increase the number of unit tests within the system. What is also interesting about this data, aside from the bump from 0 to 20 tests present in the system, is perhaps the fact that some unit tests were created outside the time frame of the planned iterations. G-Unit was available to the developers outside the planned iterations simply because the authors felt the tool should be available to the developers if they felt they benefited from its use in their daily work.

Figure 6.1a shows test sessions posted by the 3 developers from the beginning of the first iteration to the end of the last iteration. The 3 developers posted 154 test sessions to the gamification server in total.



(a) Test sessions over time



(b) Unit tests over time

Figure 6.1: Quantitative data

6.4.2 Badges

The developers had the possibility of earning 4 different badges in the system. Out of those badges, two different badges were earned by the developers, the Test-A-Day badge, and the Bronze Badge. The Test-A-Day badge was earned 11 times among the developers, and the Bronze Badge was earned once by a single developer. It is likely that the other two badges, The Silver and Gold Badge, were too hard to acquire by the developers, as they required them to write 25 and 50 tests, respectively.

6.5 Revisiting the Research Questions

6.5.1 RQ1: How does gamification influence unit testing practices?

The data indicates that gamification has a positive effect on unit testing. The gamification elements that were used in this study add a competitive element to unit testing which motivated the subjects to become more engaged in the gamified tasks. This resulted in increased testing activity in the form of more unit tests written and test sessions. The participating developers were able to learn about various unit testing metrics and concepts through G-Unit and that testing knowledge will aid them in further projects.

6.5.2 RQ1.1: How does gamification affect the developers' motivation to write more unit tests?

Both qualitative and quantitative data implies that gamification increases the motivation to write more tests. The gamification system that was created in this study seems to, in retrospect, have favoured quantity rather than quality and thus the results could be influenced by that fact. The reasons for that are several, namely that introducing quantitative metrics is easier than quality metrics and gamification systems have a harder time in general measuring quality than quantity. This is further discussed in Chapter 7.

6.5.3 RQ1.2: How do test smells and code coverage as input to a gamification system influence unit test quality?

The code coverage statistics had a positive impact on unit tests quality. The display of the code coverage statistics enabled the subjects to visualize which parts of the system had been tested and which parts of the system needed further test work. The subjects focused on writing more tests instead of raising their code coverage percentages, which is similar to the findings of RQ 1.1. Again, this is the result of the design of the gamification system which enabled the subjects to gather points more easily through writing more tests instead of raising the code coverage of old tests. Test smell results were not as positive as the code coverage results. Due to unfortunate circumstances the developers were unable to utilize the test smell detection system to its full potential, both due to developer participation issues and a possible bug within the test smell detection. However, the subjects felt that the test smell detection was useful, but not as useful as the code coverage statistics. This is further discussed in Chapter 7.

6.5.4 RQ2: What gamification element in this study proves to be the most effective at influencing unit testing practices?

The badges were the least effective element in this study. The reasons might be that badges in general are not as motivating as the other elements (leaderboard, point system) or that the design of the badge system was unbalanced. Further work is needed to answer that. The leaderboard was the most effective element in this study. The reason is that it adds a competitive element to unit testing, which motivated the subjects to become more engaged and increase their testing activity. Although the subjects reported that the leaderboard was the most effective there is a possibility that the point system is just as effective, because the leaderboard is a visualization of the point system. The leaderboard caused no or minimal ethical issues in the study and the ethical impact of the leaderboard was lesser than expected. This is heavily related to the fact that the leaderboard was not monitored by management at the case company.

7

Discussion

7.1 Research Questions

RQ1: How does gamification influence unit testing practices? The method of applying gamification to unit testing practices depends heavily on the context it is applied in, but as discussed further in this section, gamification seems to influence unit testing practices positively. This aligns with Ivarsson and Johansson's positive findings on the gamification of unit testing [10]. These findings are also similar to the findings of Hamari et al. [9], which reported increased engagement of gamified tasks. In Advania's case, there were three participating developers. Unit testing practices and gamification were introduced into their then already-established software development process, where their product has already shipped and their focus was shifting from new feature development to maintenance and support. Even though the developers had not put much emphasis on unit testing previously, they were open to this change and felt motivation from the gamification elements to write unit tests. The gamification system added an element of competition to their unit testing practices. The developers also learned about the various unit testing-related such as test smells through the tool, although they reported that they wanted more information regarding code coverage. Even though the developers adapted to having the gamification system added to their development process, the authors saw that it was important to also adapt the gamification tool to the context it was placed in. For instance, the rules in the first iteration concerning total code coverage awarded points for every 10% increment of coverage achieved by the tests. This turned out to be completely unreachable for the developers due to the size of their system and the number of tests needed to reach such a high coverage percentage.

RQ1.1: How does gamification affect the developers' motivation to write more unit tests? According to the developer feedback collected over the course of the 3 iterations performed, the developers were positive towards gamification of unit testing. During the interviews the developers felt that gamification was both an innovative and effective way to deal with motivational problems in the context of software testing, especially in regards to test quantity. This is also backed up by the daily survey data that was collected during all iterations. On almost all occasions in the surveys did they feel that the gamification system motivated them to write more tests, but were more sceptical towards the idea that the gamification resulted in higher quality tests. This adheres to previous research on gamification, as Shahri's research implies that gamification systems in general have a harder time measuring quality than quantity [21]. They were not opposed to the idea that gamification could result in higher quality tests but rather they found the technical challenge of finding appropriate metrics to evaluate the tests complex. They agreed that the G-Unit tool as a whole motivated them to write more tests, especially the leaderboard. Therefore the authors conclude that gamification can positively influence testing activity in the context of the participating company and the influence on quantity is bigger than on quality. This is also supported by the fact that the developers would like to continue the use of G-Unit or parts of G-Unit.

RQ1.2: How do test smells and code coverage as input to a gamification system influence unit test quality? The feedback from the developers suggest that the code coverage information provided by the tool had a positive influence on unit test quality. During the interviews the developers felt that the gamification system provided them with a good overview of the work of other developers, which is similar to the findings of Passos et al.[20] research on gamification systems. The developers agreed that the gamification system somewhat inspired them to raise their code coverage percentages, but did not feel as motivated to raise their code coverage compared to motivation to write more tests. The authors attribute this to the fact that the gamification system allocated a large amount of points for the first test created each day. The fact that the code coverage reward system was a bit unbalanced in the first iteration, which was the most active iteration, could also have influenced these findings. The test smell detection did not yield as positive results, for a variety of reasons. First off, the test smell detection was not implemented until iteration 2. After the test smell detection had been implemented it became apparent that the developer that was on vacation for iteration 2 and 3 had the highest amount of test smells and therefore was never able to act upon the test smells. Also, the developer that was the most active during the iterations never received warnings that his test code contained any test smells. Due to these circumstances the test smell detection was one of the least successful part of the study. These problems might have been circumvented if the G-Unit tool had been introduced during the start-up phase of the Inna project, i.e. the project the developers were working on at Advania, instead of in the maintenance phase of the project. There is also a possibility that the test smell detection was somewhat flawed because the authors expected much higher frequency of test smells within the system. These findings on the impact of test smell detection are disappointing, but are the result of the unmanageable circumstances mentioned above. Researchers should not let these findings hinder them on further research of test smells, their use in gamification systems, and their effect on unit test quality.

RQ2: What gamification element in this study proves to be the most effective at influencing unit testing practices? This study had three gamification elements, a point system, a leaderboard which displays the score of the developers and five different badges. Although the developers reported both through surveys and interviews that all three elements affected their work, both the qualitative and the quantitative data seems to support that the badges were the least effective element. All the developers acquired the test-a-day badge, which was awarded when they submitted a new unit test that day. However, only one developer acquired the bronze badge which was awarded for writing a total amount of 10 unit tests. The developer that acquired that badge was able to do so on the very last day of the last iteration. The authors feel that this indicates that the badges provide some positive effects on the developers work, but not to a great extent. This adheres to Johansson's and Ivarsson's [10] findings on the effectiveness of badges. This is also supported by the fact that during the interviews the developers noted that the leaderboard was the largest motivator to write more tests. However, the authors must recognize the fact that the badge system might have been unbalanced. In order to acquire the bronze badge a developer had to write 10 tests, to acquire the silver badge a developer had to write 25 tests and a gold badge was awarded after 50 tests. If the scale would have been significantly lower the developers might have been more interested in acquiring the badges as they could have felt overwhelmed by the amount of tests that they needed to write in order to reach the silver and bronze badge. The interviews revealed that the leaderboard was notably the biggest impact factor of the gamification system. Both of the interviewed developers noted that the competitive nature of the leaderboard was the strongest motivational factor during the study. This is in line with Singer & Schneider's experimental results, where they used a leaderboard to encourage source control commits among their students. In their work, they noted that the competitive nature of the leaderboard encouraged the students to act more in-line with Singer & Schneider's target behaviour [19].

Although the developers did not want to end up in the last place within the leaderboard they did not feel the stress to be overwhelming, but rather just the right amount to keep them on their toes. However, they both noted that if management would have monitored the leaderboard the stress levels could potentially become toxic to their work environment. These findings are perfectly in line with Shahri's findings on the ethics of gamification and Shari's instructions on how management should manage the leaderboard [21]. It is hard to evaluate if the leaderboard or the point system are more important here, since the leaderboard is a visual representation of the point system. A leaderboard built upon an unbalanced score system could therefore be ineffective or in worst case toxic to the work environment, since players within an unbalanced system could end up spending time on unimportant things and thus have a bad influence on the project as a whole. During the interviews the subjects felt that the score system served its purpose despite of its early flaws. They also felt the score system was an effective way to increase testing activity.

7.2 Contribution to Knowledge

This study contributes to knowledge within the field of software engineering in various ways.

TestHound Maven Plugin
Although Greiler et al. had plans to release TestHound as a Maven plugin, no plugin was available at the time this report is written. This study therefore extends Greiler's work on on automatic detection of test smells by making G-Units' TestHound Maven plugin available on GitHub. [1]. This plugin could aid practitioners who are interested in measuring their unit test quality while researchers could use the plugin in research related to automatic quality analysis of test code.

Gamification & Software Development This thesis was largely inspired by Johansson & Ivarsson's experiment [10] on the gamification of unit testing. In their future work chapter they recommend moving gamification from a controlled environment and into an organizational context which is precisely what this study does. The authors did not find many studies that study gamification of the software development cycle, except for Singer [19] and Passos [20]. This study therefore contributes to the body of knowledge in terms of gamification of the software development cycle as previous research in that field is relatively limited.

Creating a gamification system based on both internal (test smells) and external (code coverage) quality metrics of unit tests is a challenging task. The G-Unit tool score system could serve as a model for other researchers interested in gamifying unit testing. However, the score system is by no means perfect and would need fine tuning when the context is changed.

7.3 Validity Threats

This section covers validity threats based on the definition of validity threats provided by Runeson and Höst [38].

Construct Validity: The evaluation of the iterations was performed by questionnaires

[1]https://github.com/davidarnarsson/GUnit/tree/master/gunit-maven-plugin

and interviews. The developers answered daily questionnaires during each iteration and a post-iteration questionnaire. There is a possibility that the daily questionnaires were annoying to complete. Developers answering the question might have answered the questionnaires without thinking about their answers simply to get the task of answering daily questionnaires out the way. There is also a possibility that the developers might be biased in their feedback since they might have answered the questions based on what they feel is acceptable or desired by the author. To counter this threat the questionnaires were designed with the guidelines provided by [38] Runeson and Höst in mind.

The participating developers in the study are Icelandic but have a solid background in the English language as well. Therefore it was decided to provide the developers with the questionnaires both in Icelandic and English since some technical terms don't have a direct translation in Icelandic.

Internal Validity: No steps had been taken prior to the study by the participating department to increase unit testing or other kinds of automated testing. Although the quantitative data shows more testing activity after the introduction of the tool and qualitative data suggests that the developers find the tool motivating to write more tests it must be acknowledged that the developers might be writing more tests because the company accepted to take part in the study and therefore the developers felt pressured to write more tests. This still remains a possibility even though the developers reported no pressure from the company to increase testing when asked about this validity threat during the evaluation of iteration 3. The test smell detection system might have contained hidden bugs that the authors were unable to locate, since the developer with the highest amounts of tests written had no detectable smells. Theoretically, his tests might have contained no test smells at all, but a more likely reason is that the test smell detection system was somewhat flawed. However, the detection system seems to have been at least partially working, since it detected test smells in the test code of the other two developers.

External Validity: The study was conducted at only one company with varying number (1-3) of participating developers. The evaluation of the G-Unit tool and the gamification system are specific to that one company and the results of this study are not generalizeable. However, this study could serve as a basis for future work in the field. Over time, researchers could either replicate this study, expand it or perform the study on a larger population to achieve statistical power. Researchers interested in unit testing, test smells, gamification of the software development cycle or gamification in general could borrow elements from this study and expand on these topics in their work.

Reliability: One of the authors is an employee of the company where the study was conducted. The author worked in web development within the same company, although in a different web development division than where the study was conducted. Therefore the authors had a certain advantage over other researchers that would be interested in replicating the study since the authors were familiar with the company. However this advantage is not that big of an impact factor for the study, although it made the initial problem investigation and communication with management easier. Other researchers that have a solid technical background in Java development and some experience in unit testing should be able to take this thesis along with the source code from GitHub (the code is open source) and replicate the study easily, given a similar context. The bigger issue here would be replicating the study in a different context. This study relied heavily on the TestHound tool for smell detection, which works exclusively with Java. If a researcher was interested in replicating the study he could run in to problems regarding test smell detection, if the researcher isn't working in Java,x as there aren't many tools available that detect flaws within unit test source code. A researcher would probably have to create a smell detection tool himself, which is quite a technical challenge. The other main concepts, gamification and code coverage, are easier to deal with in a different context. There are many different code coverage tools available online and literature on gamification is easily accessible in the form of books and scientific articles.

8

Conclusion and future work

This study examined the influence of gamification on unit testing practices in industry. The gamification tool introduced to Advania had an effect on the company in the form of increased testing activity by all the participating developers and the developers learnt about testing concepts and metrics, such as code coverage and test smells. This acquired testing knowledge will aid the developers in their future projects. The ethical issues related to the application of gamification within organizations were not observed to a high degree, which is probably due to the fact that there were only 3 subjects in the study and management did not monitor the leaderboard. Although the results were positive in most cases, if not all, measures could be taken to further enhance the effects. Notably, more subjects participating in the study and launching the tool at project start would have been ideal, but unfortunately that was not an option in this case. As anticipated, the balancing of the gamification system and its metrics proved to be the largest design problem, while integrating test smell detection was the biggest technical challenge. Creating a motivating gamification system is a challenging task and metrics have to be chosen wisely in order to achieve the predefined goals. Gamification in general is a useful tool to solve motivational problems, but as with all tools, has to by applied correctly.

8.1 Future work

The authors would like to encourage other researchers to expand upon this study by conducting an experiment or a case study within an organization that measures the impact of gamification on either unit testing or another testing method. Unit testing was chosen in this study as the authors are most experienced in that field of testing. However, the authors feel that another testing method could easily be gamified given that sensible metrics are available for that method. Although this study did not encounter the negative ethical aspects of gamification to a high degree, possibly because the number of developers participating in this study was relatively low, other researchers should pay special attention to the ethical aspects and methods that mitigate the risks related to the those aspects. Other tasks of the software development cycle could also be interesting to gamify. For instance, the authors feel that continuous integration or continuous deployment could easily be gamified, for example by rewarding developers for frequent releases and/or frequent integration. Another idea would be to conduct an experiment where researchers focus on which aspects of gamification are the most motivating, for example by comparing the effects of a leaderboard with the effects of badges and/or achievements. A word of advice to researchers that are interested in gamification, be it in software development or not, is that balancing a score system within a gamification system is a complex task and players within the system will often immediately try to find ways to game and/or cheat the system. For example the authors encountered this problem within the first hour after deploying G-Unit. With regards to unit testing, the authors feel that good results and more developer participation could be achieved if a tool such as G-Unit would be introduced to new software development right from the start, in particular when paired with a methodology which features testing as a core practice. It would also be interesting to see the a study done where there are more participating developers. Losing a developer due to vacations or illness could possibly affect the other developers' motivation to participate. There were only 3 developers participating in this study, which means that any such losses have a more pronounced effect than not. Having more developers could alleviate such effects.

The TestHound Maven plugin could also be expanded upon by other researchers. For example, researchers could add other test smell detection tools to the plugin and thereby extend the plugin's ability to detect faults within test code. Practitioners could integrate the plugin to their code base as a first step to evaluate their companies unit test quality. Researchers and practitioners interested in extending this plugin are advised to read up on van Deursen's work regarding test smells [5] and Greiler's et al. article on TestHound [29].

Appendices

A

Appendix

A.1 Interview guide: Code coverage in Iceland

- What is your opinion on code coverage, in the context of your software company?
- Do you currently employ code coverage as a metric of unit test suite quality?
- Any specific goals you strive for, with regards to code coverage?

A.2 Interview Guide: Iteration 3 post-interview

- The tool
  - Do you think it helps with writing better tests? (If yes, then how?)
  - Do you think it helps with writing more tests? (If yes, then how?)
  - Do you think you want to continue using it, or some parts of it, after the iterations?
- The process
  - Do you feel that the gamification system motivated you to to write more tests? (If yes, then how?)

  - Do you feel that the gamification system motivated you to write higher quality tests? (If yes, then how?)
  - Do you think that gamification is applicable to unit testing in industry?
  - Do you think you will continue to write
- Motivation
  - Are you writing tests because of the tool or because we and/or the company are forcing you to do so?

A.3 Gamification rules

A.3.1 Iteration 1

Bronze Badge Rule

Awarded when the total amount of tests written by a user reaches 10 tests. Given that the test suite covered in Advania's case is very small, the authors felt that 10 tests warranted a bronze badge.

Silver Badge Rule

Similar to the gold and bronze rules above, the silver badge is awarded to a user for writing 25 tests.

Gold Badge Rule

Similar to the bronze badge rule, the gold badge is awarded when the total amount of tests written by a user reaches 50 tests.

Branch Coverage Rule

Awards points for increasing the project-wide branch coverage in 10% increments. The rule also deducts points for decreasing the branch coverage, which commonly occurs when there have been additions to the code base. Deducted points can then in turn be re-earned by adding tests for the newly added code. Up to 15 points for 60% or higher branch coverage can be awarded.

Instruction Coverage Rule

Similar to the branch coverage rule, the instruction coverage rule awards or deducts points depending on the total instruction coverage reached by a given user. The points awarded depends on the instruction coverage reached. For example, 5 points are awarded for reaching 50% instruction coverage.

Test-A-Day Rule

The Test-A-Day rule awards a user 20 points and a badge for submitting a new test each day. The badge and points are only awarded once each day.

A.3.2 Iteration 2

In addition to all of the rules specified in iteration 1, iteration 2 added the following rules:

Branch Coverage Per Class Rule

The branch coverage per class rule awards or deducts points the branch coverage reach per class with branch coverage >0%. The maximum amount of points per class given is 5 points for 100% branch coverage. The number of available points are scaled per class depending on the total median number of branches per class. The minimum amount of points given for a class with 100% coverage is 1 points.

So, for n classes the total amount of points if given by

$$\sum_{c=1}^{n} max(minPoints, round(min(maxPoints, classCoverage_c/medianCoverage*maxPoints))) \tag{A.1}$$

Surpassing User Rule

The surpassing user rule does not award any points to a user. It generates notifications directed towards a user that just surpassed other users on the leaderboard.

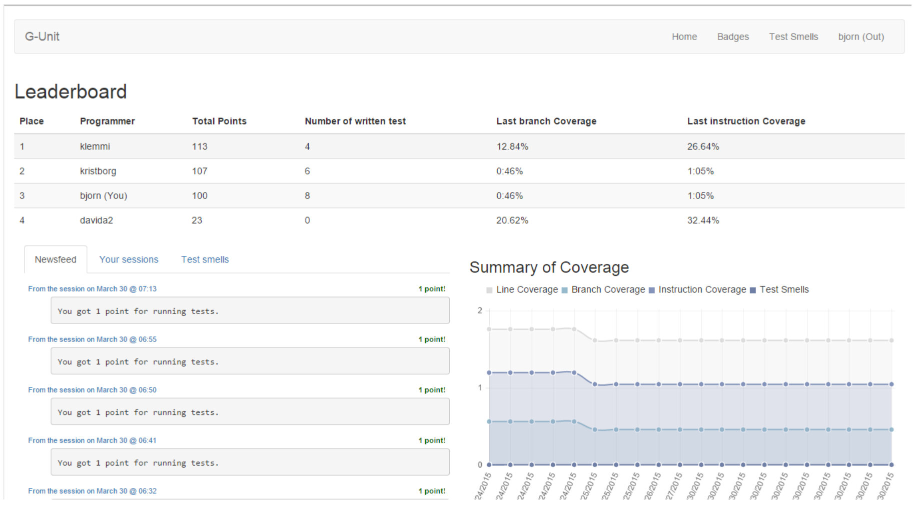
Test Smells Rule

The test smells rule awards points for any test smells, previously detected, that were refactored by the user. A point is awarded for each test smell refactored.

Execution Rule

Awards a single point every time the entire test suite is executed.

A.4 Pictures of G-Unit



Leaderboard

Place	Programmer	Total Points	Number of written test	Last branch Coverage	Last instruction Coverage
1	klemmi	113	4	12.84%	26.64%
2	kristborg	107	6	0:46%	1:05%
3	bjorn (You)	100	8	0:46%	1:05%
4	davida2	23	0	20.62%	32.44%

From the session on March 30 @ 07:13 1 point!

```
You got 1 point for running tests.
```

From the session on March 30 @ 06:55 1 point!

```
You got 1 point for running tests.
```

From the session on March 30 @ 06:50 1 point!

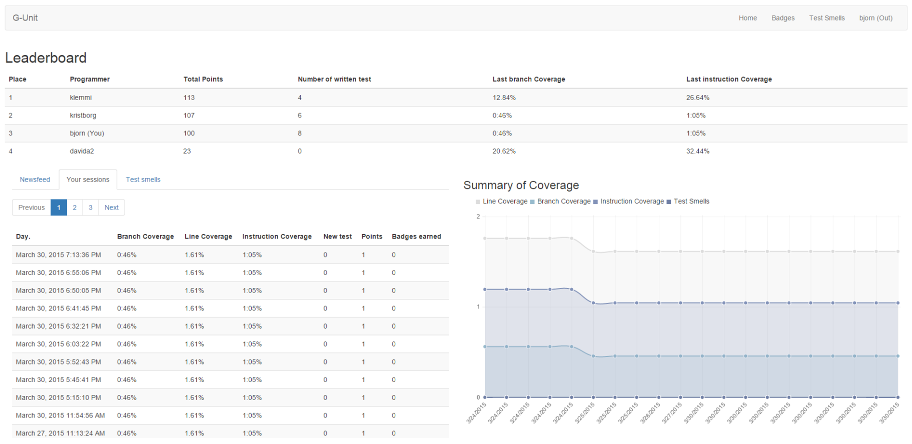
```
You got 1 point for running tests.
```

From the session on March 30 @ 06:41 1 point!

```
You got 1 point for running tests.
```

From the session on March 30 @ 06:32 1 point!

Figure A.1: Front page - News feed

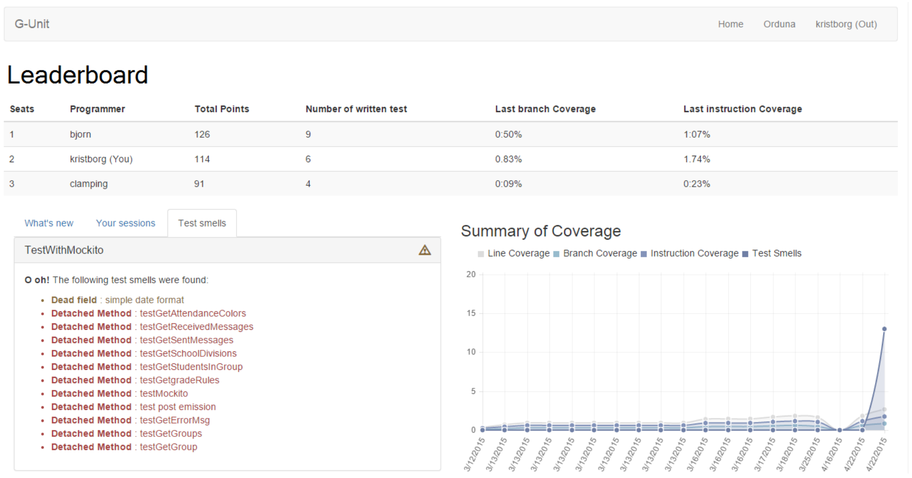


Leaderboard

Place	Programmer	Total Points	Number of written test	Last branch Coverage	Last instruction Coverage
1	klemmi	113	4	12.84%	26.64%
2	kristborg	107	6	0:46%	1:05%
3	bjorn (You)	100	8	0:46%	1:05%
4	davida2	23	0	20.62%	32.44%

Day.	Branch Coverage	Line Coverage	Instruction Coverage	New test	Points	Badges earned
March 30, 2015 7:13:36 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 6:55:06 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 6:50:05 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 6:41:45 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 6:32:21 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 6:03:22 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 5:52:43 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 5:45:41 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 5:15:10 PM	0:46%	1.61%	1:05%	0	1	0
March 30, 2015 11:54:56 AM	0:46%	1.61%	1:05%	0	1	0
March 27, 2015 11:13:24 AM	0:46%	1.61%	1:05%	0	1	0

Figure A.2: Front page - Sessions



Leaderboard

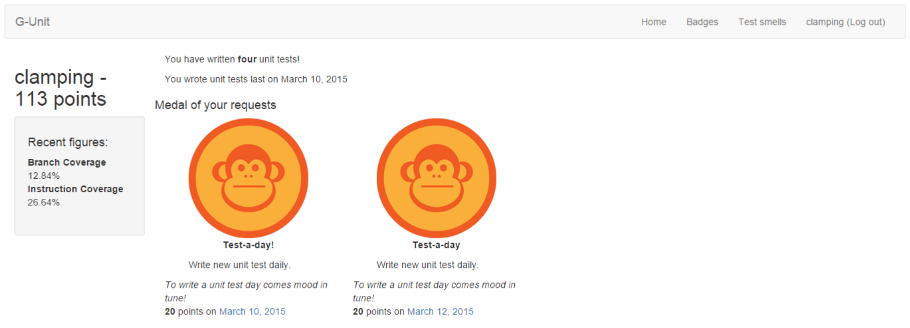
Seats	Programmer	Total Points	Number of written test	Last branch Coverage	Last instruction Coverage
1	bjorn	126	9	0:50%	1:07%
2	kristborg (You)	114	6	0.83%	1.74%
3	clamping	91	4	0:09%	0:23%

TestWithMockito

O oh! The following test smells were found:

- **Dead field** : simple date format
- **Detached Method** : testGetAttendanceColors
- **Detached Method** : testGetReceivedMessages
- **Detached Method** : testGetSentMessages
- **Detached Method** : testGetSchoolDivisions
- **Detached Method** : testGetStudentsInGroup
- **Detached Method** : testGetgradeRules
- **Detached Method** : testMockito
- **Detached Method** : test post emission
- **Detached Method** : testGetErrorMsg
- **Detached Method** : testGetGroups
- **Detached Method** : testGetGroup

Figure A.3: Front page - test smells



clamping - 113 points

Recent figures:
Branch Coverage
12.84%
Instruction Coverage
26.64%

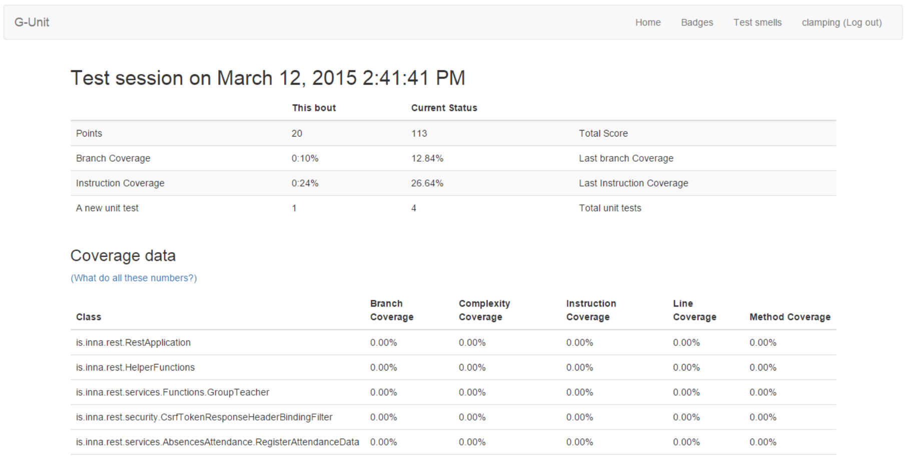
You have written **four** unit tests!
You wrote unit tests last on March 10, 2015

Medal of your requests

Test-a-day!
Write new unit test daily.
To write a unit test day comes mood in tune!
20 points on March 10, 2015

Test-a-day
Write new unit test daily.
To write a unit test day comes mood in tune!
20 points on March 12, 2015

Figure A.4: User profile page



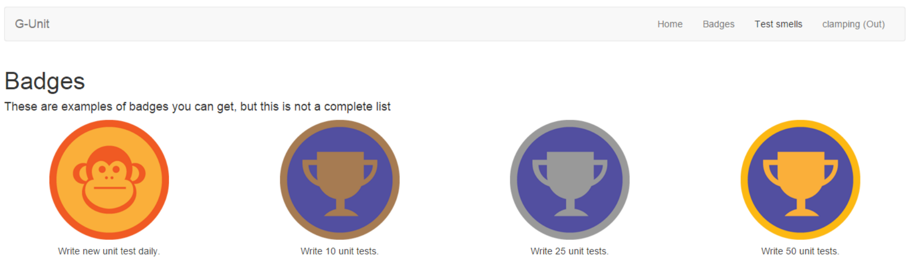
Test session on March 12, 2015 2:41:41 PM

	This bout	Current Status	
Points	20	113	Total Score
Branch Coverage	0:10%	12.84%	Last branch Coverage
Instruction Coverage	0:24%	26.64%	Last Instruction Coverage
A new unit test	1	4	Total unit tests

Coverage data
(What do all these numbers?)

Class	Branch Coverage	Complexity Coverage	Instruction Coverage	Line Coverage	Method Coverage
is.inna.rest.RestApplication	0.00%	0.00%	0.00%	0.00%	0.00%
is.inna.rest.HelperFunctions	0.00%	0.00%	0.00%	0.00%	0.00%
is.inna.rest.services.Functions.GroupTeacher	0.00%	0.00%	0.00%	0.00%	0.00%
is.inna.rest.security.CsrfTokenResponseHeaderBindingFilter	0.00%	0.00%	0.00%	0.00%	0.00%
is.inna.rest.services.AbsencesAttendance.RegisterAttendanceData	0.00%	0.00%	0.00%	0.00%	0.00%

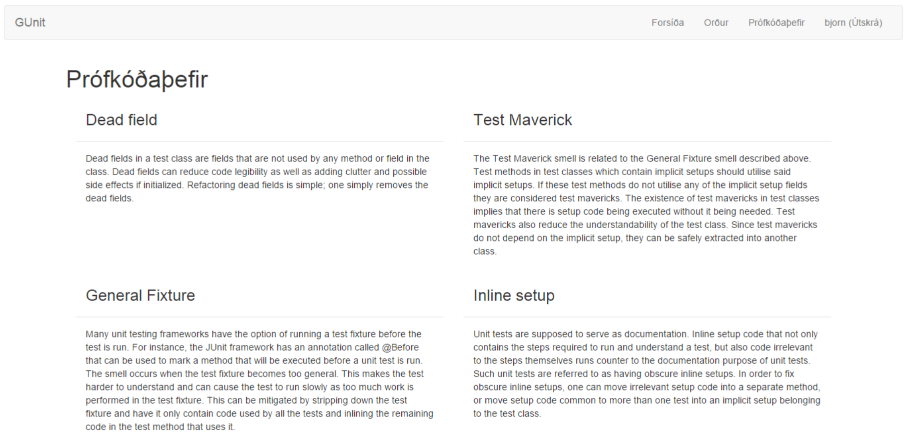
Figure A.5: Test session page



Badges

These are examples of badges you can get, but this is not a complete list

Figure A.6: Badges page



Prófkóðaþefir

Dead field

Dead fields in a test class are fields that are not used by any method or field in the class. Dead fields can reduce code legibility as well as adding clutter and possible side effects if initialized. Refactoring dead fields is simple; one simply removes the dead fields.

Test Maverick

The Test Maverick smell is related to the General Fixture smell described above. Test methods in test classes which contain implicit setups should utilise said implicit setups. If these test methods do not utilise any of the implicit setup fields they are considered test mavericks. The existence of test mavericks in test classes implies that there is setup code being executed without it being needed. Test mavericks also reduce the understandability of the test class. Since test mavericks do not depend on the implicit setup, they can be safely extracted into another class.

General Fixture

Many unit testing frameworks have the option of running a test fixture before the test is run. For instance, the JUnit framework has an annotation called @Before that can be used to mark a method that will be executed before a unit test is run. The smell occurs when the test fixture becomes too general. This makes the test harder to understand and can cause the test to run slowly as too much work is performed in the test fixture. This can be mitigated by stripping down the test fixture and have it only contain code used by all the tests and inlining the remaining code in the test method that uses it.

Inline setup

Unit tests are supposed to serve as documentation. Inline setup code that not only contains the steps required to run and understand a test, but also code irrelevant to the steps themselves runs counter to the documentation purpose of unit tests. Such unit tests are referred to as having obscure inline setups. In order to fix obscure inline setups, one can move irrelevant setup code into a separate method, or move setup code common to more than one test into an implicit setup belonging to the test class.

Figure A.7: Test smells page

B

Questionnaires

In this section, the questionnaires performed in this study are presented, along with the data collected from each questionnaire.

B.1 Before introduction

This questionnaire was sent to the developers before the introduction of G-Unit.

1. How interested were you in writing unit tests today?
   - ☐ Very little interest
   - ☐ Little interest
   - ☐ Neither disinterested nor interested
   - ☐ Interested
   - ☐ Very interested
2. How much motivation did you feel in regards to writing unit tests today?
   - ☐ Very weak motivation
   - ☐ Weak motivation
   - ☐ Neither strong nor weak motivation
   - ☐ Strong motivation
   - ☐ Very strong motivation

3. Which of the following describes your opinion on unit tests?
   - ☐ Very useless
   - ☐ Useless
   - ☐ Neither useless nor useful
   - ☐ Useful
   - ☐ Very useful

4. How fun is it to write unit tests?
   - ☐ Very boring
   - ☐ Boring
   - ☐ Neither fun nor boring
   - ☐ Fun
   - ☐ Very fun

5. "I think that unit tests take too much time from the development of the actual product" How strongly do you agree with this statement?
   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly agree

B.1.1 Data

Timestamp	Question 1	Question 2	Question 3	Question 4	Question 5
3/10/2015 14:40:10	Interested	Weak motivation	Useful	Neither fun nor boring	Neutral
3/10/2015 14:43:52	Interested	Very weak motivation	Very useful	Neither fun nor boring	Strongly disagree
3/11/2015 10:00:26	Very little interest	Very weak motivation	Useless	Very boring	Strongly agree

B.2 Developer daily: Iteration 1

This questionnaire was sent to the developers every day during iteration 1.

1. How interested were you in writing unit tests today?
   - ☐ Very little interest
   - ☐ Little interest
   - ☐ Neither disinterested nor interested
   - ☐ Interested
   - ☐ Very interested
2. How much motivation did you feel in regards to writing unit tests today?
   - ☐ Very weak motivation
   - ☐ Weak motivation
   - ☐ Neither strong nor weak motivation
   - ☐ Strong motivation
   - ☐ Very strong motivation
3. How fun was it to write unit tests today?
   - ☐ Very boring
   - ☐ Boring
   - ☐ Neither fun nor boring
   - ☐ Fun
   - ☐ Very fun
4. "I think that unit tests take too much time from the development of the actual product" How strongly do you agree with this statement?
   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly agree

5. How hard was it to use the test analzying tool? Mark the choice that best describes your opinion.
   - ☐ Very hard
   - ☐ Hard
   - ☐ Neither easy nor hard
   - ☐ Easy
   - ☐ Very easy
6. Did you feel that the point system motivated you to write more unit tests?
   - ☐ Yes
   - ☐ No
7. Did you feel that the point system motivated you to write better unit tests?
   - ☐ Yes
   - ☐ No
8. Did you feel uncomfortable that your points are compared to the points of other programmers within the leaderboard?
   - ☐ Yes
   - ☐ No

B.2.1 Data

Timestamp	Question 1	Question 2	Question 3	Question 4	Question 5	Question 6	Question 7	Question 8
3/12/2015 16:00:11	Very interested	Neither strong nor weak motivation	Neither fun nor boring	Strongly disagree	Neither easy nor hard	Yes	Yes	No
3/12/2015 17:53:12	Interested	Neither strong nor weak motivation	Fun	Disagree	Very Easy	Yes	No	No
3/13/2015 9:15:38	Neither disinterested nor interested	Weak motivation	Boring	Agree	Easy	Yes	No	No
3/13/2015 15:39:24	Interested	Strong motivation	Neither fun nor boring	Strongly disagree	Easy	Yes	Yes	No
3/13/2015 15:40:43	Neither disinterested nor interested	Strong motivation	Fun	Disagree	Easy	Yes	Yes	No
3/17/2015 9:53:04	Little interest	Neither strong nor weak motivation	Neither fun nor boring	Agree	Easy	Yes	No	No
3/17/2015 10:12:17	Neither disinterested nor interested	Neither strong nor weak motivation	Neither fun nor boring	Strongly disagree	Easy	Yes	Yes	No
3/17/2015 16:33:44	Little interest	Neither strong nor weak motivation	Boring	Agree	Easy	Yes	No	No
3/17/2015 18:33:44	Neither disinterested nor interested	Strong motivation	Neither fun nor boring	Neutral	Easy	Yes	Yes	No
3/18/2015 16:14:06	Little interest	Weak motivation	Neither fun nor boring	Strongly disagree	Very Easy	No	No	No
3/19/2015 17:19:53	Little interest	Neither strong nor weak motivation	Boring	Agree	Easy	No	No	Yes
3/19/2015 17:20:13	Interested	Neither strong nor weak motivation	Fun	Disagree	Easy	Yes	Yes	No

B.3 Post-Iteration 1

An open-ended questionnaire was sent to the developer post-iteration 1.

1. What improvements to the leaderboard would you make, if any? ____________________
2. Would you rather that the leaderboard assigns you a discrete rank instead of continuous points?
   ☐ Yes
   ☐ No
3. What improvements to the badge system would you make, if any? ____________________
4. Any comments regarding the tool? ____________________

B.3.1 Data

The data from the post-iteration 1 questionnaire can be found in Table 6.1.

B.4 Developer daily: Iteration 2

1. How interested were you in writing unit tests today?
   ☐ Very little interest
   ☐ Little interest
   ☐ Neither disinterested nor interested
   ☐ Interested
   ☐ Very interested
2. How much motivation did you feel in regards to writing unit tests today?
   ☐ Very weak motivation
   ☐ Weak motivation
   ☐ Neither strong nor weak motivation
   ☐ Strong motivation
   ☐ Very strong motivation

3. How fun was it to write unit tests today?
   - ☐ Very boring
   - ☐ Boring
   - ☐ Neither fun nor boring
   - ☐ Fun
   - ☐ Very fun
4. “I think that unit tests take too much time from the development of the actual product” How strongly do you agree with this statement?
   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly agree
5. How hard was it to use the test analzying tool? Mark the choice that best describes your opinion.
   - ☐ Very hard
   - ☐ Hard
   - ☐ Neither easy nor hard
   - ☐ Easy
   - ☐ Very easy
6. Did you feel that the point system motivated you to write more unit tests?
   - ☐ Yes
   - ☐ No
7. Did you feel that the point system motivated you to write better unit tests?
   - ☐ Yes
   - ☐ No
8. Did you feel that the test analyzing tool motivated you write better unit tests?
   - ☐ Yes
   - ☐ No
9. Did you feel uncomfortable that your points are compared to the points of other programmers within the leaderboard?

☐ Yes
☐ No

B.4.1 Data

Timestamp	Question 1	Question 2	Question 3	Question 4	Question 5	Question 6	Question 7	Question 8	Question 9
4/22/2015 17:25:21	Interested	Neither strong nor weak motivation	Fun	Disagree	Easy	Yes	Yes	Yes	No
4/25/2015 13:56:32	Neither disinterested nor interested	Neither strong nor weak motivation	Neither fun nor boring	Disagree	Neither easy nor hard	Yes	Yes	Yes	No
4/27/2015 16:18:49	Neither disinterested nor interested	Neither strong nor weak motivation	Neither fun nor boring	Disagree	Easy	Yes	Yes	Yes	No
4/28/2015 15:03:52	Little interest	Weak motivation	Neither fun nor boring	Disagree	Easy	Yes	Yes	Yes	No

B.5 Post-iteration 2

1. I understand what test smells are.
   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly agree
2. I understand in what ways test smells can affect my test suite.
   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly agree
3. In what ways could the test smells display be improved, if any? ____________________
4. I think the "Newsfeed" gives me enough information regarding the scoring system.
   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly agree
5. In what ways could the "Newsfeed" display be improved, if any? ____________________
6. I think that seeing what badges are available motivates me to write more tests.
   - ☐ Strongly disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly agree
7. Any general comments? ____________________

B.5.1 Data

Timestamp	Question 1	Question 2	Question 3	Question 4	Question 5	Question 6	Question 7
4/30/2015 12:00:22	Agree	Agree		Agree		Agree	

Bibliography

[1] G. J. Myers, C. Sandler, T. Badgett, The art of software testing, John Wiley & Sons, 2011.

[2] E. M. Maximilien, L. Williams, Assessing test-driven development at ibm, in: Software Engineering, 2003. Proceedings. 25th International Conference on, IEEE, 2003, pp. 564–569.

[3] K. A. Briski, C. Poonam, V. Hamilton, A. Pratt, B. Starr, J. Veroulis, B. Villard, Minimizing code defects to improve software quality and lower development costs, IBM Rational Software Analyzer and IBM Rational PurifyPlus software.IBM, 2003.

[4] S. McConnell, Code complete, Microsoft press, 2004.

[5] A. van Deursen, L. Moonen, A. van den Bergh, G. Kok, Refactoring test code, CWI, 2001.

[6] M. Fowler, Refactoring: improving the design of existing code, Pearson Education India, 2002.

[7] K. Huotari, J. Hamari, Defining gamification: a service marketing perspective, in: Proceeding of the 16th International Academic MindTrek Conference, ACM, 2012, pp. 17–22.

[8] S. Deterding, D. Dixon, R. Khaled, L. Nacke, From game design elements to gamefulness: defining gamification, in: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, ACM, 2011, pp. 9–15.

[9] J. Hamari, J. Koivisto, H. Sarsa, Does gamification work?–a literature review of empirical studies on gamification, in: System Sciences (HICSS), 2014 47th Hawaii International Conference on, IEEE, 2014, pp. 3025–3034.

[10] M. Johansson, E. Ivarsson, An experiment on the effectiveness of unit testing when introducing gamification, Master's thesis, Chalmers University of Technology (June 2014).

[11] Y. K. Malaiya, N. Li, J. Bieman, R. Karcich, B. Skibbe, The relationship between test coverage and reliability, in: Software Reliability Engineering, 1994. Proceedings., 5th International Symposium on, IEEE, 1994, pp. 186–195.

[12] H. Þrastar Björnsson, Skype interview, Reon Tech, conducted: 15 Mar. 2015.

[13] H. B. Olafsson, Skype interview, Plain Vanilla Games, conducted: 15 Mar. 2015.

[14] T. Kanij, R. Merkel, J. Grundy, An empirical study of the effects of personality on software testing, in: Software Engineering Education and Training (CSEE&T), 2013 IEEE 26th Conference on, IEEE, 2013, pp. 239–248.

[15] L. De-Marcos, A. Domínguez, J. Saenz-de Navarrete, C. Pagés, An empirical study comparing gamification and social networking on e-learning, Computers & Education 75 (2014) 82–91.

[16] Video games revenue worldwide from 2012 to 2015, `http://www.statista.com/statistics/278181/video-games-revenue-worldwide-from-2012-to-2015-by-source/`, accessed: 2015-03-03.

[17] A. Marczewski, Gamification: a simple introduction, Andrzej Marczewski, 2012.

[18] S. Grant, B. Betts, Encouraging user behaviour with achievements: an empirical study, in: Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on, IEEE, 2013, pp. 65–68.

[19] L. Singer, K. Schneider, It was a bit of a race: Gamification of version control, in: Games and Software Engineering (GAS), 2012 2nd International Workshop on, IEEE, 2012, pp. 5–8.

[20] E. B. Passos, D. B. Medeiros, P. A. Neto, E. W. G. Clua, Turning real-world software development into a game, in: Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on, IEEE, 2011, pp. 260–269.

[21] A. Shahri, M. Hosseini, K. Phalp, J. Taylor, R. Ali, Towards a code of ethics for gamification at enterprise, in: The Practice of Enterprise Modeling, Springer, 2014, pp. 235–245.

[22] Apache Foundation, "maven – welcome to apache maven." maven – welcome to apache maven, `http://maven.apache.org/`, accessed: 17 Apr. 2015.

[23] D. Martin, J. Rooksby, M. Rouncefield, I. Sommerville, 'good'organisational reasons for'bad'software testing: An ethnographic study of testing in a small software company, in: Software Engineering, 2007. ICSE 2007. 29th International Conference on, IEEE, 2007, pp. 602–611.

[24] G. Tassey, The economic impacts of inadequate infrastructure for software testing, National Institute of Standards and Technology, RTI Project 7007 (011).

[25] N. Juristo, A. M. Moreno, S. Vegas, Reviewing 25 years of testing technique experiments, Empirical Software Engineering 9 (1-2) (2004) 7–44.

[26] J. A. Whittaker, What is software testing? and why is it so hard?, Software, IEEE 17 (1) (2000) 70–79.

[27] Y. Cheon, G. T. Leavens, A simple and practical approach to unit testing: The jml and junit way, in: ECOOP 2002—Object-Oriented Programming, Springer, 2002, pp. 231–255.

[28] S. Cornett, Code coverage analysis, Bullseye Testing Technology.

[29] M. Greiler, A. van Deursen, M.-A. Storey, Automated detection of test fixture strategies and smells, in: Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on, IEEE, 2013, pp. 322–331.

[30] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, D. Binkley, An empirical analysis of the distribution of unit test smells and their impact on software maintenance, in: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, IEEE, 2012, pp. 56–65.

[31] R. Wieringa, Design science as nested problem solving, in: Proceedings of the 4th international conference on design science research in information systems and technology, ACM, 2009, p. 8.

[32] A. R. Hevner, A three cycle view of design science research, Scandinavian journal of information systems 19 (2) (2007) 4.

[33] S. Chatterjee, B. Tulu, T. Abhichandani, H. Li, Sip-based enterprise converged networks for voice/video-over-ip: implementation and evaluation of components, Selected Areas in Communications, IEEE Journal on 23 (10) (2005) 1921–1933.

[34] K. Peffers, T. Tuunanen, M. A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, Journal of management information systems 24 (3) (2007) 45–77.

[35] J. Masters, The history of action research, Action research.

[36] W.-C. Leung, How to design a questionnaire, student BMJ 9 (11) (2001) 187–189.

[37] S. E. Hove, B. Anda, Experiences from conducting semi-structured interviews in empirical software engineering research, in: Software Metrics, 2005. 11th IEEE International Symposium, IEEE, 2005, pp. 10–pp.

[38] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical software engineering 14 (2) (2009) 131–164.