The relation between software testing, reliability and work well-being

Master's Thesis
University of Turku
Department of Future Technologies
Software engineering
June 2019
Sami Nieminen

Supervisors:
Tuomas Mäkilä
Anne-Maarit Majanoja

UNIVERSITY OF TURKU
Department of Future Technologies

SAMI NIEMINEN: The relation between software testing, reliability and work well-being

Master's Thesis, 52 p., 13 app. p.
Software engineering
June 2019

Over the last decades, software development has evolved from slow and tedious to swift and agile process which has enabled the easy and customer-centric development of new applications. Being able to interact wirelessly with almost any household item or share memories and media with shorter latencies than ever raises the importance of proper and safe software. The software development principles are not the same they were even twenty years ago but instead they keep constantly evolving and industries come up with better ways to deliver exceptional products for a wider and more demanding audience.

Testing is unquestionably one of the most valuable factors of software development which unfortunately is too often neglected and forgotten. Many organizations struggle to keep their code maintainable which in the worst case can have devastating consequences risking the whole business. Tests are the key factors that can when done properly, reflect the quality of the software and could even increase productivity.

Work environment, happiness, and well-being during the development phase could massively influence the quality of the software. This thesis explores the possible connection between well-being and software development and attempts to find the answer to whether work well-being could be improved solely by proper testing.

To study this claim a questionnaire was conducted by collecting responses from software developers with different backgrounds and experiences. The goal of this thesis was to produce a valuable insight to software developers and companies about the phenomenon which can possibly be used to improve the quality of life of workers and software. The results indicate that different types and number of testing tools do have a connection on how an individual feel about their work and projects.

The data would hopefully encourage other researches to perform more profound studies around the topic but also to promote companies and individual developers to evaluate their tools and processes.

Keywords: Testing, software reliability, work well-being, software engineering, questionnaire

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos

SAMI NIEMINEN: The relation between software testing, reliability and work well-being

Diplomityö, 52 s., 13 liites.
Ohjelmistotekniikka
Kesäkuu 2019

Ohjelmistokehitys sekä -käytännöt ovat viimeisen parin vuosikymmenen aikana muuttuneet radikaalisti hitaista ja vaativista, nopeisiin sekä ketteriin menetelmiin. Tämä muutos on näkynyt yhä enemmän muun muassa asiakaslähtöisessä sekä helpommin lähestyttävässä kehitystyössä. Nykyisten huippunopeiden sekä langattomien verkkojen myötä ohjelmistot ovat yhä laajemmassa käytössä ja samalla niiden laadun sekä luotettavuuden vaatimukset ovat korostuneet.

Testaus on kiistämättä yksi ohjelmistokehityksen tärkeimmistä osa-alueista, mutta sen hyödyntäminen osana kehitysprosessia on valitettavan usein laiminlyöty tai jätetty tekemättä kokonaan. Hyvin järjestetyn testauksen avulla voidaan saavuttaa suuria hyötyjä, kuten nopeuttaa kehitystyötä sekä parantaa ohjelmiston luotettavuutta. Nämä ovat tärkeitä ominaisuuksia, joilla ohjelmistoyritykset voivat erottua toisistaan.

Työympäristö, onnellisuus sekä työhyvinvointi kehitystyön aikana voivat vaikuttaa huomattavasti ohjelmiston laatuun. Tämän tutkielman tavoitteena oli tutkia, että voidaanko ohjelmistokehittäjien työhyvinvoinnin, käytettyjen testaus- sekä kehityskäytäntöjen välille löytää yhteys, josta saataisiin tietoa menetelmistä, joilla tuotantoa saataisiin tehostettua.

Tutkimusta varten laadittiin verkkokysely, jonka avulla kerättiin tietoja ohjelmistokehittäjien työhyvinvoinnista, testauskäytännöistä sekä järjestelmien luotettavuudesta. Tuloksista nähdään, että käytettävillä menetelmillä ja työkaluilla on eroja siihen, miten työntekijät kokevat henkiset voimavaransa sekä kuinka miellyttävää työskentely on.

Asiasanat: Ohjelmistotestaus, luotettavuus, työhyvinvointi, ohjelmistotuotanto, kyselytutkimus

Contents

1 Introduction **1**

2 Software testing **3**

2.1 Testing 4

2.2 Development processes 5

2.2.1 Agile 6

2.3 Agile testing 8

2.3.1 Extreme Programming 9

2.3.2 Test Driven Development 10

2.3.3 Behavior Driven Development 11

2.3.4 Acceptance Test Driven Development 11

2.4 Automated tests 12

2.4.1 Unit tests 13

2.4.2 Integration tests 13

2.4.3 System tests 14

2.4.4 Acceptance tests 14

2.4.5 Other 14

3 Software reliability in quality analysis **16**

3.1 Measuring software quality 17

3.1.1 Software quality models 18

3.2 Reliability 21
3.2.1 Quality management standards 22
3.3 Quality Assurance 23
3.4 Security 23
3.5 Code coverage 24

4 Work well-being **25**
4.1 Measuring work well-being 26

5 Methods **30**
5.1 Objectives 31
5.2 Target group 31
5.3 Data collection 32
5.4 Analysis 33

6 Survey **34**
6.1 Cover letter structure 34
6.2 Topics 35
6.3 Questions 36
6.3.1 Work well-being 36
6.3.2 Testing tools 36
6.3.3 Software Reliability 37
6.3.4 Demographics 38
6.3.5 Feedback and contact information 38

7 Results **39**
7.1 Overview 39
7.2 Comparison 46
7.3 Discussion 50

8 Conclusion **51**

References **53**

Appendices

A Questionnaire **A-1**

B Questionnaire schema **B-1**

List of Figures

2.1 Project management triangle 5

2.2 Waterfall model 6

2.3 Scrum process 8

2.4 Testing pyramid 13

7.1 Correlation of well-being and stress answers per question 43

7.2 Result how fulfilling developing current project is. 47

7.3 Satisfaction and experience. 48

7.4 Code reviews and work fulfillness. 49

7.5 Amount of testing tools and sense of satisfaction. 49

List of Tables

3.1 McCall's quality factors 20

4.1 Four-Factor Model of Workplace Well-Being 27
4.2 Components of enduring success 28

7.1 Responses by position 40
7.2 Responses by experience 41
7.3 Responses by organization size 41
7.4 Responses by organization type 41
7.5 Question 5: Please, estimate following claims concerning your software testing. 44
7.6 Question 6: Please select all of the applicable categories. Our organization has a dedicated tool, which manages the following testing aspect. . . 45
7.7 Question 7: Please, estimate following claims concerning problems about testing. 46
7.8 Question 8: Please, estimate following claims concerning your current (or latest) software project. 47

B.1 Questionnaire well-being and stress schema B-1

1 Introduction

Software organizations have noticed a need for more efficient processes to keep up with growing competition and a more demanding audience. Modern software development processes e.g. agile have guided the development in the right direction where not only the code and tools are evaluated continuously but also the focus is kept on improving the working environment [1]. This change has caused the development projects success rates to improve vastly [2].

After test-automation emerged along with agile software development the whole industry has realized the importance of testing and how it can have a huge impact on project length. Often competition is won by the companies that can deliver promises to the customer the fastest [3].

Lately increasing number of research has been done on finding factors for a happy life and occupational well-being which is thought to be due to a rapid global industrial growth [4]. Some researchers suggest that there is a noticeable connection between job satisfaction and performance [1]. Despite the wide range of studies, there are not many focusing on the software industry or on how the stress and mental health affects the development process. Software development is a constant race between time and costs and combined with requirements it is crucial to have a working development processes and employees who are able to work at their best.

The objective of the thesis is to find a correlation between work well-being, software testing, and reliability. The assumption is that adapting high-quality software testing pro-

cesses or tools to a development workflow can have an impact on an individual's personal experience towards work. Software reliability is examined as a how effortless and comfortable process it is to add new features, make releases and develop a software product overall. The assumption is that developers and testers perform better when they can trust the software they're building, and that this can improve the productivity of an organization as well as an individual developer.

Chapter 2 is an introduction to modern software testing principles, metrics, and tools. Chapter 3.2 walks the reader through different software quality models and popular processes. Chapter 4 focuses on work well-being generally. To find answers to the hypotheses an online questionnaire is conducted that includes several questions concerning each topic. The methods used for collecting the questionnaire responses, analysing the results and objectives are listed in chapter 5 and the structure of the questionnaire is explained thoroughly in chapter 6. Chapter 7 presents the findings and comparison of the results to existing research as well as the discussion. Conclusions and further research are discussed in chapter 8.

2 Software testing

The end goal in software development is to maximize quality and minimize costs. There are as many ways of developing software as there are developers but keeping the code maintainable, meaningful, and extendable require a lot more experience than just making it run. There are many cases of companies with highly successful and popular products who were brought down because they were not able to keep up with the customer demand because of rushing in the development phase and neglecting error reports. Poor system design choices and non-existing tests are often the results of an unsustainable code base with high technical debt. [5]

Even though the developers who are really talented, experienced and have the know-how to implement all features and requirements of the product owner, the code which is written in such manner that it is not thought of being passed forward for the next developer, will fail or at least might became the avoided part of the code base [5]. There is no absolute and unique way of creating code which could live forever. Instead there are many principles that can reinforce the software development process and guide towards "clean" manageable code [5].

Software development processes in the 1950s differed a lot from modern equivalents. The initial assumption was that the completed code will always contain errors that should be fixed quickly and continue to the next task. This basically meant that there was no room for stopping to think about design choices. Instead, making the decisions were supposed to be done along as the code progressed [6]. One can imagine how it must have felt like a

new programmer starting on such a project. Methods have improved a lot since.

Getting familiar and trusting the tools within the working environment can lead to immense improvements for both the overall development as well as the happiness of the team. Being able to communicate with the team and finding problems fast is crucial on agile development. [3]

2.1 Testing

Testing is a highly important part of the software development process. Being able to write tests straight from the beginning makes the code more prone to updates and bugs. Testing methods have improved a lot from the early days of programming and more tools are introduced to help to automate the processes. Proper tooling and habits lead to faster execution times, better analytics and more fluent reliable shipping of software. [3]

The worst enemy of the software developer is most often time. The project management triangle (Fig.2.1) describes that a project can be done by selecting only two of the properties but it is not possible to have the best quality done with low expenses in little time. Software testing can make achieving the corners bit more by strengthening developers trust in the project. The aim with software testing is to validate the accuracy of logic but also adhere to functional requirements which brings the value to the software. Tools have improved a lot over the years and whenever new programming languages or paradigms have emerged, testing also evolves along with it. Both the software industry and academia studies automated testing and companies have moved towards continuous integration tools to improve workflow even further. [3]

Modern problems are more often solved by utilizing online resources. The ongoing shift towards utilizing the internet has introduced new ways of writing software for example in a microservice manner. Distributing parts of the whole application mean new challenges for the testing tools. [3] The whole industry has come a long way from building

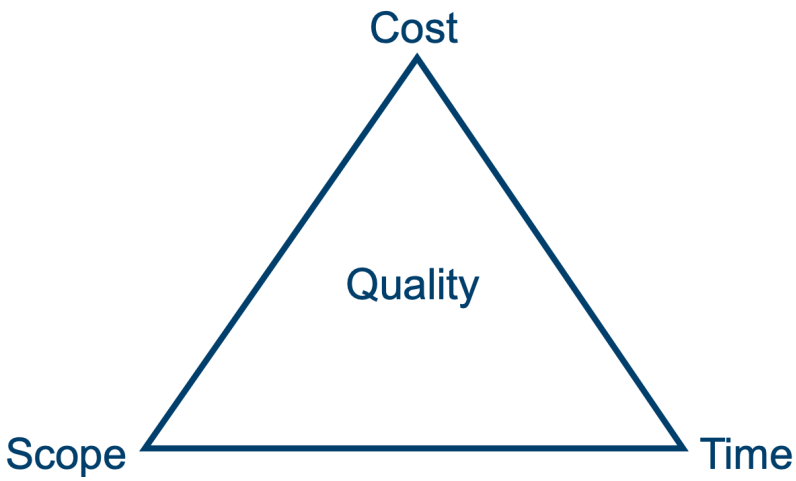

Figure 2.1: Project management triangle

large entities continuously for weeks or even months with late iterations and tested only retroactively, to small and short steps where a developer or tester can make the decision of iteration instead of sitting through meetings with the product owners.

2.2 Development processes

Software development consists of different guidelines and lifecycles which are often characterized as development processes. These guidelines describe the roadmap how a company can develop software to ensure high quality, efficiency, and fulfil customer's demand. For keeping the software quality high it is common for companies to follow, adapts or combine some of the common software development processes but processes are useless unless the organization and teams accept the rules and follow them. High process quality is almost necessary for creating high-quality software. [6]

One of the first widely recognized development processes was the *Waterfall model* (figure 2.2). In the model, each step has a single input and output meaning the process could advance only after the previous step is completed. The process starts by analysing the requirements as thoroughly as possible to form a specification. After the specification is completed documents are passed forward to the design step where they are analysed

Figure 2.2: Waterfall model

to determine overall system architecture, required hardware, etc. The requirements are then divided into different teams or developers who work individually on the task. When developers are satisfied with the product it is passed forward to integration and system testing phase. [7].

Combining multiple individually long time developed entities into one single testable package often fails and causes delays due to certain section must be renewed completely [7]. The waterfall model is known to have a tendency to go over budget but also to fail completely [2].

In the waterfall model there often were no tooling or processes for automating tests or evaluating the quality of code. That meant all tests were done manually and only when bugs were encountered [3]. Other popular development processes were the spiral model, the Rational Unified Process and the Agile methodology [6]. The scope of this thesis is on the new Agile methodology and how it has guided the development towards tests first thinking.

2.2.1 Agile

Agile software development is an umbrella term for different frameworks that were created to dismiss old heavy processes like the waterfall model and to follow the same principles [8], [6]. The Agile methodology urges to evaluate current development in small

cycles which makes it possible to notice and discard bad design choices right from the beginning. It also emphasizes the importance of working in teams, incremental development and adaptive planning. [6].

In the early 1990s companies had realized that old development methods are not working for them and there is a need for a lighter and more efficient way of building software [8]. Studies show that projects which were using the waterfall model, in the halfway through the project 50% of the features of 100% of the requirements were fulfilled where the projects using Agile methods had met 100% of the features of 50% of the requirements [6]. This simply states that the agile way of doing emphasizes to have something always working where the traditional methods, on the other hand, relied upon that everything is ready at the same time in the end.

In the 1990s there was not any consistent way of describing these new kinds of development processes. The term "Agile" was not conducted until the 2001 when 17 software developers, e.g. Robert Martin, Martin Fowler and the inventor of Extreme Programming (XP) Kent Beck, who have being part of developing frameworks and practices, dedicated to improve the software development processes, gathered around to discuss about the methodologies and they authored the Agile Manifest [8]. The manifest consists of 12 principles which can be simplified to [8]:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Scrum project management framework is one agile adaptation where the development happens in sprints, varying from one day to full month. Tasks are piled into storage as known as *product backlog* and into it is stored any future ideas or error reports, etc. Tasks

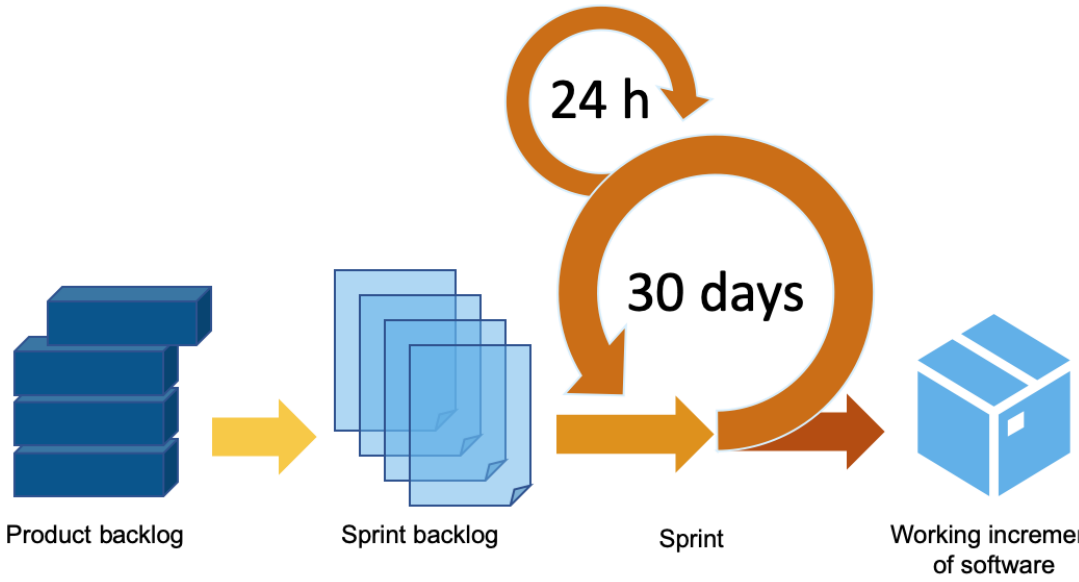

Figure 2.3: Scrum process

are then evaluated on daily or weekly meetings with the development team where only necessary and suitable tasks are selected into a *sprint backlog*. The spring backlog is then emptied during the next sprint along with continuous testing and evaluations. After each sprint progress is evaluated and the goal is to always have something working. The process is illustrated in Fig 2.3.

2.3 Agile testing

Agile Manifesto started a movement which then rapidly displaced the more traditional waterfall and many other hefty models. Along with short development cycles the importance of testing increased and that has sprouted many new types of development processes [6]. Along with high connectivity through internet usage and increasing pressure from the customer side has shortened the shipping cycles when at the same time the requirements have increased [9].

Traditionally a software development process has been divided into two-team two-step process where one of the teams has been responsible for producing the code and

another testing it. After the developer team has been satisfied with the product it is passed along to the testers. Agile methods encourage performing development and testing in close connection but also in a cooperative manner with potential testing or development teams. This way the testing can achieve higher quality and a lot faster response time for handling bug reports as well as moving to work on new features [3]. Projects success rates conducted by Agile teams have been proven to outperform traditional Waterfall projects. [2]

2.3.1 Extreme Programming

The Agile Manifesto created a concept around existing rules and practices on how to develop software more efficiently and with less drag. One of the most popular methodologies is Extreme Programming (XP) mentioned in the last chapter. The XP is basically a set of Agile practices but targeted more for a single developer to follow.[10]

Main practices can roughly be divided into two levels:

- Developer level
  - Test Driven Development
  - Continuous Integration
  - Refactoring
- Team level
  - Acceptance Testing
  - User stories
  - Short release cycles
  - Small releases

The developer level practices are easy to implement and can almost be used by an individual even though the whole team or business might not use XP. These are easy

practices to implement and helps a developer to see feedback instantly, therefore these are often the first steps towards applying the whole XP methodology into the development workflow.[10]

The team level practices, on the other hand, require a lot more not only from the team but also from an individual because they require an interaction with the customers and often with people outside of the team. [10]

2.3.2 Test Driven Development

Test Driven Development (TDD) practice was introduced in the early 21st century by developer Kent Beck. The key concept of TDD is to write tests before writing actual code [5]. The process starts by evaluating requirements and converting them to unit tests. Because new features are not yet implemented, initially every test should fail and this works as a basis on which the software or parts of it are built around. On the first cycle the only goal is to make new tests pass. Writing the code fast does not always lead to the fully optimal and clean end result but the next step after new features are working on themselves is to ensure them to be in unison with the existing code base and features. These steps might require refactoring and migrations but more importantly refining the quality of newly written code. [6] By writing tests to match the requirements right from the beginning developers are notified immediately if a refactoring breaks something and one or many tests fail. This way developers are always aware what is affected by latest change. [5]

Three laws of TDD [5]:

- You may not write production code until you have written a failing unit test
- You may not write more of a unit test than is sufficient to fail, and not compiling is failing
- You may not write more production code than is sufficient to pass the currently

failing test

Following these rules ensures that programs are developed in short enough cycles and because of tests and production code are written at the same time catching breaking changes can be seen immediately. This way it would be possible to achieve 100% code coverage which means that every line and every branch is checked by some unit test. In reality this method can be a really time-consuming process and any larger changes automatically mean tens or even more broken tests. [5]

2.3.3 Behavior Driven Development

Behavior-driven development (BDD) is an agile technique that aims to fill the gap between developers and stakeholders to ensure that the product is built correctly right from the beginning. It introduces so-called *feature* files which are written in natural language that can be read and written by not programming-oriented people but still be computable by computers e.g. *Gherkin* for the *Cucumber* testing tool. In BDD features are written down with or by the customer into *features* which developers can convert into test cases. [11]

2.3.4 Acceptance Test Driven Development

Acceptance Test-Driven Development (ATDD) is the actual process of running automated tests defined with the BDD process. Customers own and define tests which make it possible to verify user stories as accurate as possible. By letting the customer be part of the testing and whole software development process can help to notice important features quicker and more accurately opposite of developers writing tests that they think are essential. Due to its customer-centric nature acceptance tests are sometimes called as customer tests. [10]

User stories are descriptions of requirements divided into short texts by features. A popular approach is writing requirements on index cards or sticky notes which are easy

to reorganize and visualize on a table, wall or on software. The method how the notes are organized does not matter, but the important thing is to attain a clear channel between the reviewer and the customer which helps to prioritize tasks. [6]

After definition user stories are then converted into tests by the developers or by the customers themselves. By engaging customers to be part of the development and splitting required features into smaller parts can lead to more thought out cases and therefore moving the workload of from the developers.[10]

2.4 Automated tests

For being able to maintain rapid development cycles and save valuable time tests also needs to cause little friction as possible. With modern testing libraries and test runners, most of the tests can be automated and be integrated as part of the development and can be performed by each individual developer without the need of additional tester. [9]

Automated tests are more reliable, faster and cheaper than traditional manual testing because when written once tests can be shared as part of the software via version control system so other members can run same tests individually. A single member does not need to memorize all the options and views to test when tasks are automated. Manual testing can introduce more bugs and are easier to be dismissed or modified accidentally [9].

A common representation of kind and ratio between the testing types is the test pyramid (Fig.2.4). It states that most of the tests should be unit tests that build a solid base for other tests to rely on [12]. On the next step integration tests are responsible for evaluating the operation of component interaction of two units or a set of modules performing larger tasks. The top section acceptance tests are to verify programs function as a whole and that it matches the requirements. Pyramid also illustrates well what is the cost of development on each level. Unit tests are small and fast to write because they concentrate on small sections of code which are visible for a single developer. Integration tests require

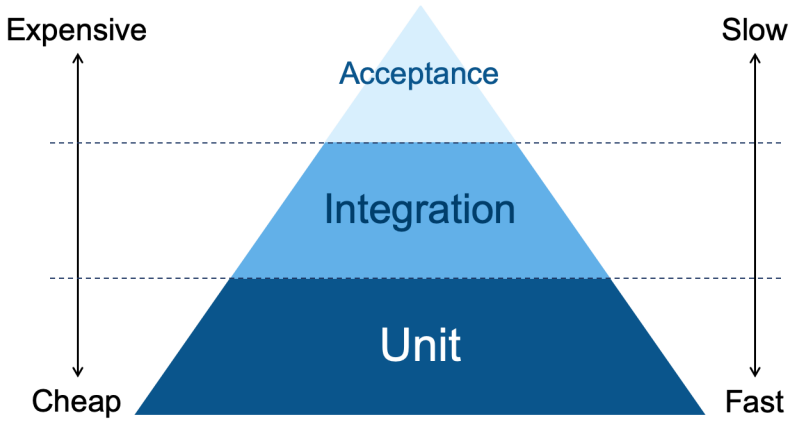

Figure 2.4: Testing pyramid

more understanding of the underlying codebase and about the connection between units. Then the size and focus of acceptance tests then should be discussed with the customer, which will inevitably increase the costs and requires time [12][13]. These testing types are explained in finer detail on the next chapters.

2.4.1 Unit tests

Unit testing means testing the basic unit of software where the *unit* could be a single function, module or component with one mission [3]. Most of the software tests should be a unit test because they are fast to run, simple to write and they guides towards cleaner and more manageable code [5]. Secondly they make debugging easier because in case of an error the exact module can be pinpointed directly [9].

2.4.2 Integration tests

Integration tests are testing the integration of two or more single units or modules of the software as one. Integration testing should be started after all the unit tests are passing [6]. The end goal is to verify that all the components and their interfaces are fully functional together [3].

2.4.3 System tests

System tests are meant for verifying the software as a whole and that the implementation has been done in such fashion that everything matches the given requirements. System tests are often conducted by different people than the ones on developer teams, who report inconsistencies and bugs back to the developers. [6]

2.4.4 Acceptance tests

Acceptance tests are the basic block of ATDD, the process mentioned in the previous chapter (2.3.4). Test cases are defined by the customer and are afterward converted into actual tests to the software domain language. Acceptance tests give an indication about user acceptance and confidence but are also a good way of keeping the communication channel open between a customer and a team. Whereas unit tests focus on covering most of the software on a low level, acceptance tests are more often presenting wider features of the whole system and might intersect with system tests.[3]

2.4.5 Other

UI testing

Tracking visual changes of an application has always been rather difficult due it often consists of multiple abstraction levels like in web development displaying something on the screen requires HTML for structuring, CSS for styling elements and JavaScript for creating possible feedback from the user action. Some early UI testing implementations relied on taking pictures and it was developers responsibility to spot the differences. Nowadays there are plenty of libraries and programs to ease the pain by automating the process and storing snapshots of the combined output.

Static testing

Static testing differs from other automated and dynamic testing so that the code under evaluation is not executed but instead the environment or external programs scan the code for possible defects. Type checking in Java or TypeScript in JavaScript are examples of static analysis of the code. In statically typed languages static tests are run automatically either directly or during a build time which is great for catching typos and type errors which otherwise would be caught up only after compiling. Also manually going through the code or performing code reviews are considered as static testing. [14]

3 Software reliability in quality analysis

The software development community has created many different models describing ways of creating streamline and fully functional programs but still, none of them has been successfully carried out every aspect of software quality perfectly [6]. This chapter describes what are the key aspects of software quality, what are the tools for measuring it and popular models and principles like the *quality management system* (QMS).

Generally, quality can be understood as something being good or bad in the task which it is designed for e.g, a musical instrument could be said to be bad in quality if it cannot play music or it does not keep its tune. The same basic principle can be applied to software as well as any other product or service. Before modern software development frameworks and globally available internet, the focus on computer programs was highly on functionality over customer's satisfaction. The trend has turned completely and towards highly monitored processes using iterative development with tightly coupled interaction with the end users and customers. [15][16]

Markets favor high-quality products that can answer the demand and are developed and improved rapidly. This enforces the importance of software quality measurements. Measuring the quality is crucial in order to keep the balance between being able to produce an acceptable product within a budget. Stakeholders i.e. developers, testers, users, customers, and managers should understand at least these metrics which also helps the whole organization to keep up continual improvements. Right choices can be done on the first try but most of the time it comes through a long process that requires a lot of trial and

error, research and development, and adapting the continual improvement strategy with the evaluative development process can grant great results. [15]

3.1 Measuring software quality

Institute of Electrical and Electronics Engineers' (IEEE) defines ([17]) a software as:

> *Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.*

The definition separates four different components: computer programs, procedures, documentation, and data to handle. For attaining maximum quality all of the four components must be present throughout the development. The first one is a computer program which in other words means the code and is rather self-exploratory why and where it is needed. Procedures describe in which order programs are executed but it does not only concern the code but also all the processes that are required in order to create the software. These include setting the responsibilities of team members as well as for example guidelines for testers. Documentation builds a story about development processes around the whole software. It contains the requirements report, design reports, program descriptions, etc. which can work as an assistant for later development or as a user manual for new developers. The last component is the data that covers all variables and parameters required to implement necessary features. The data formed during the development is the aspect which creates software's value. [18]

Quality would not exist without unintended behavior errors and in order to understand what we can measure with quality models, it is first important to recognize reasons that cause software failures. The software can contain two types of defects: software errors and software faults. Errors are mistakes done by a programmer do due to some typo a logical error or inadequate documentation not matching the requirements. Faults are possible error states that are caused by errors. Not all errors cause havoc and can be

easily fixed without any noticeable fault conditions. A system's quality or reliability is then perceived through software failures. Failures that can happen when the fault is large enough and prevents regular use. A program might be faulty but failures are never encountered and similarly, a single error can cause large problems. [18]

The understanding of common errors and software failures has guided the industry to create patterns and guidelines to prevents such defects and measured as "software quality". [18]

3.1.1 Software quality models

The inevitable fact that software will always contain some errors has guided the industry to define models that help to detect problems in the early stages and being able to improve their processes during the development. A better understanding of good practices can increase software quality. [18]

Software quality has many different definitions e.g. IEEE's definition ([17]) from 1991 described it as:

1. *The degree to which a system, component, or process meets specified requirements.*
2. *The degree to which a system, component, or process meets customer or user needs or expectations.*

McCall's model

Pursue high quality in the software development field is not a new thing. It dates back in the mid-1970s when McCall, Richards and Walters applied quality factors and criteria to software [19][15]. This led to the creation of a still popular model used for evaluating quality in the software business. Their technical report "Factors in Software Quality" [19] aimed to specify properties that can be used to measure software quality for the Air Force. The model is commonly known as *McCall's factor model tree*

The model defines 11 quality factors that present a behavioral characteristic of a software system [18][20]. Quality factors are listed on a Table 3.1. The factors are grouped into three categories: product operation, product revision and product transition. Quality in software means different things for different people and it is measured from different angles, for example, customers are looking for reliability and efficiency when developers and testers are more interested in portability and reusability which helps them to match the requirements [18].

Boehm

Barry W. Boehm's model McCall's model and it was created a couple of years after in 1978. It attempts to qualitatively measure software quality with a predefined set of metrics and attributes. The model separates quality characteristics into three levels: high, intermediate and primitive. High level represents software's from the perspective of a buyer i.e. how well it can be used, how easy it is to understand and modify, and how well it works on different environments. Intermediate level represents quality characteristics which are expected from software and these are: flexibility, understandability, efficiency, usability, testability, reliability and portability. On the primitive level product quality is assessed based on 17 factors that share reliability, usability, efficiency, maintainability and portability with McCall's definition [21]

ISO 9126

The International Organization for Standardization (ISO) is a global non-governmental international organization which is responsible for globally used standards like ISO 9001 quality management, ISO 22000 food safety management or ISO 639 language codes [22]. They are also responsible for developing more practical standards like the CD-ROMs and specifications for C++ programming language [22].

The ISO 9126 standard quality model is derived from McCall's model which includes

Quality Factors	Definition
Correctness	Extent to which a program satisfies its specifications and fulfills the user's mission objectives
Reliability	Extent to which a program can be expected to perform its intended function with required precision
Efficiency	Amount of computing resources and code required by a program to perform a function
Integrity	Extent to which access to software or data by unauthorized persons can be controlled
Usability	Effort required to learn, operate, prepare input, and interpret output of a program
Maintainability	Effort required to locate and fix a defect in an operational program
Testability	Effort required to test a program to ensure that it performs its intended functions
Flexibility	Effort required to modify an operational program
Portability	Effort required to transfer a program from one hardware and/or software environment to another
Reusability	Extent to which parts of a software system can be reused in other applications
Interoperability	Effort required to couple one system with another

Table 3.1: McCall's quality factors

more standardized measurements that help to compare products to others [21]

3.2 Reliability

Reliability is one of the most used factors when measuring software quality [15]. It is present in all quality models mentioned in the last section. Software's reliability can be defined as a probability of encountering errors within a predefined time, often expressed as the mean time to failures (MTTF). Software, as it is immaterial does not suffer from any physical interaction like the hardware but instead, it is either working or broken from the start. This emphasizes the importance of good practices during the design and development of phases [6].

Reliability is often inspected more from the user's perspective by how well something operates. A program is considered more reliable if it repeats tasks consistently without errors, opposite to the one which fails to perform the task, has long delays, displays error messages or crashes frequently. *Market windows* is a concept of a time interval available to how long software is reasonable to develop before the cost of the process exceeds possible market value or the elapsed time of development to produce high quality and reliable software is surpassed by competitive product. This window is hard to predict but companies need to be conscious about their surroundings and be willing to make trade-offs between reliability and cost to meet the requirements in the schedule. [15]

System requirements are crucial for determining software reliability but in real-world applications, it is impossible to cover all possible use cases in reasonable time and cost. Properly written requirements can guide the development well e.g in TDD, the developers or testers have knowledge about the most important features and where to start. When requirements change or new features have added the risk of faults and instability increases. The amount of additions to the code base is inversely proportional to the reliability. Every new feature can add instability to the system if not carefully inspected and therefore

smaller changes are easier to test comparatively to tests of multiple larger changes at once. [15]

It is difficult to create fully comprehensive tests to a large or even mid-sized software covering every possible component interactions and all possible inputs [6]. Tests increase the value and reliability of the application but even then, the profound meaning of tests is to find, remove and prevent errors from ever happening on production. The reliability can be measured high even without having a hundred percent coverage of every file, branch, and step of the code base but it requires smart and well-thought processes that will save time eventually. Testing should be thought of as the process of finding errors instead of testing existing pieces of code. By starting development with the assumption that code will contain errors helps to guide testing towards those not familiar weak-spots instead of only focusing tests on parts that are already working [9]

Projects might have some quality criteria that need to be analysed before the product or update can be published. By setting targets e.g. 100% of tests have to pass or that code base cannot include linting errors throughout the development processes team as well as the management can make assumptions about past and present quality but probably not to prevent future defects Software reliability models will help to define processes which helps avoid common pitfalls. [6].

3.2.1 Quality management standards

The ISO/IEC 90003:2018 is a set of quality management and certification standards for computer software development and maintenance organizations [18]. The ISO 9001 is a part of ISO 9000 quality management standard family and was updated last in the 2015. The Quality management system (QMS) is set of specific requirements which includes guidelines to helps companies and organizations to manage, improve and monitor product development, research and services generally and it can be applied to any size organization regardless of the size or field of study. QMS does not directly offer

specific rules how the requirements from the customers or stakeholders should be met but instead guides companies to take actions of defining these themselves. By having well-structured processes for following customer satisfaction closely can lead to increase in business [22]. These frameworks aims to help organizations to understand common concepts and principles as well as the vocabulary of the quality management in the scope of software development and therefore can increase effectiveness [23].

3.3 Quality Assurance

Quality Assurance (QA) is any process or process which guides the development process to be more visible and more maintainable for the whole organization [6]. QA is part of the *quality management system* of the ISO 9001. The goal is to make processes easy to monitor and modify which then can be standardized to maximize efficiency. [18]:

QA is performed to sustain high quality throughout the whole research and development process [16]. The Institute of Electrical and Electronics Engineers' (IEEE) definition of *quality assurances* is [18]:

1. A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.

2. A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with quality control.

3.4 Security

Most of the modern software is somehow connected to the internet but as it opens lots of possibilities like real-time updates and communication it also exposes a whole new surface for attackers to attempt to intercept messages or insert malicious code which can directly affect company's reputation and even stocks [24].

High connectivity leads to an increasing need for having the application working around the clock and regardless of the location. The software therefore also needs to be dependable. Dependability in this context means

-Security acts in a big role of creating reliable software -high quality software which can to identify severe problems in the early stages and make corrections fast to prevent such flaws from ever happening on production [18].

3.5 Code coverage

Code coverage, also referred to as test coverage is a generated report of how many and which lines of code were executed when tests were run [25]. Code coverage is often used as a metric for unit tests. An umbrella term of such testing where testers have access to the source code so they can examine the code or at least be familiar with the logic of the program and which outputs metrics to help with analyzing code quality is called white-box testing [9]. Code coverage gives an indication of how comprehensive the unit tests are. Even though code coverage varies tremendously depending on the programming language it still is widely used as a good indicator of trust towards the software. [26]

4 Work well-being

The field of work well-being is not new and it has been studied for almost a century. Mental health and happiness are more and more taken into consideration when talking about work well-being. Still, researchers and philosophers have to this day struggled to achieve a common consent about the definition of well-being [1]. The attention has shifted from measuring the absence of stress, mental illnesses and suffering as metrics of work wellness more towards the importance of mental health and employees' personal wellness in the organization. The research is often divided roughly into three overlapping categories: the hedonic well-being i.e. "experiencing a pleasant life", eudaemonic well-being i.e. "living a good life" and into social well-being which is more of an intersection between hedonic and eudaemonic views. Social well-being works in a way as a bridge between the hedonic pursuit of pleasure and eudaemonic pursuit for happiness. These views are not completely exclusive but they are more or less seen individual topics of study. [1]

Well-being is important for both the individual themselves and for the organization. There have been many studies that reinforce the assumption that happy employees actually improve well-being at work [1]. There are indications that it is easier for satisfied employees to cooperate with co-workers, be more punctual, have less sick leaves, and have a higher commitment towards the organization than the ones who are less happy with the surrounding situation or position [1]. These same studies have shown a noticeable connection between job satisfaction and performance. By having basic needs fulfilled

and a job that is challenging enough people can perform better and be more productive. [1]

There can be seen a connection between health psychology and seminal work. Findings indicate that employees who work in an environment that do not offer a variation on workload, are unsatisfactory, have low or none control over the job has more health concerns. Even though satisfaction at the workplace is widely studied, most of the measures are suffering from limitations e.g. being too lengthy, too narrow or lacking validity.

Parker's and Hyett's study of "Measurement of Well-Being in the Workplace" created a self-report measurement system to address these problems as well as previously used metrics. Their study consists of a total of 31 questions divided into four factors: work satisfaction, organizational respect for the employee, employer care and an intrusion of work into private life. These same factors are adapted and used on measuring the work well-being on this survey. [27]

4.1 Measuring work well-being

World Health Organization (WHO) defines health as: "*A state of complete physical, mental and social well-being, and not merely the absence of disease* [28]. They have also defined frames in which the organization is considered to be healthy. A healthy organization is one where workers and managers are using continual improvements process to actively improve and protect the health, safety, and well-being at work. To achieve these goals organizations have to take many things into consideration when trying to achieve a safe working environment. A safe working environment does not only mean physical space which obviously needs to be designed well to prevent accidents and have good ventilation. Psychosocial and mental health are important factors for building a healthy workplace as physical requirements. Offering a possibility to improve personal health either by exercising or occupational health services are good ways to improve overall

Factor	Description
Work Satisfaction	Does the work increase respondent's self-worth and meaning by offering enough challenges?
Respect for the Employee	How the company's senior colleagues are seen and does the organization values match the respondent's personal values.
Employer Care	Does the respondents receive enough credit from their doings and are they valued by their peers and managers.
Intrusion of Work into Private life	A negative factor which finds out how stressful the work is and that does the workload affect personal life outside the office.

Table 4.1: Four-Factor Model of Workplace Well-Being

wellness at workplace.[28]

Parker and Hyett encapsulated well-being into a model called "Four-Factor Model of Workplace Well-Being" (Table 4.1). Similarly to the WHO's definitions, the four-factor model gives a lot of weight to personal experiences and feelings.

Nash and Stevenson from Harvard business review interviewed hundreds of professionals and managers trying to find common factors for success and what keeps these people engaged with their goals and what gives them fulfilment. They separated four aspects (Table 4.2 from their studies but also concluded that it is almost impossible to maximize all of them but instead having a healthy balance. [29]

Research by Tsuneo, Yoshio and Kazuhiko [30] in 1997 found out a link between stress (mental and physical) and fault occurrences on a software project. They quantitatively measured the performance and stress tolerance of the members of two engineering teams. Teams were given the same task to build software but using a different design methodology, functional or structured. Researchers artificially created stress for the developers by changing the requirements or reducing the development time. Developers mental and physical stress were monitored using questionnaires as well as the number of faults in the program throughout the development of three months.

Component	Description
Happiness	Feelings of pleasure of contentment about your life
Achievement	Accomplishments that compare favorably against similar goals others have strived for
Significance	The sense that you've made a positive impact on people you care about
Legacy	A way to establish your values or accomplishments so as to help others find future success

Table 4.2: Components of enduring success

They concluded that mental stress does cause faults as well as physical stress i.e. catching a cold. The team which used the functional design methodology seemed to have had more mental stress but also made more faults compared to another team. They also found out that it is human nature to report fewer stress faults than really were found. [30]

The question and answer website Stack Overflow conducted a developer survey in early 2019 where they asked various questions concerning technology, developers' experiences, work, and career values but also much more. The survey received almost 90 000 responses from all over the world. Among many technical details, developers were asked about their job priorities and career values. When asked about the most important job factor more than 50% answered that the language, frameworks, and other technologies were most important. Just 6 percent behind was office environment or company culture. More people value tools over the environment. [31]

Based on the results developers are satisfied with their jobs. The survey states that more than 70% of all developers are slightly or very satisfied with their jobs when asked about their careers and jobs. Results reveal also that over 80% of respondents are very or somewhat confident about their management. These results give a promising picture of that software engineers and developers are rather happy with their current job, manage-

ment, and quality of work.[31]

5 Methods

The focus of this thesis is not to find the best tools for measuring software quality or factors that make a good workplace but instead to find a correlation between amount and type tests, subjective opinion about software reliability and well-being at work. The aim is to discover whether good testing habits as part of development workflow can improve job satisfaction of individual developer or in the best-case wellness within the whole organization. This research also attempts to answers a question that does developers and testers perform better when they can trust the software they are building.

The assumption is that untested or refactored code increases the fear of fault and failure when making releases. By letting uncertain code get into production then the customers are the ones who are finding and reporting faults this can decrease the reputation of the product or in the worst-case lead to losing customers.

The survey is being constructed by following the six principles of conducting a personal opinion survey introduced by Barbara A. Kitchenham and Shari L. Pfleeger on *Guide to Advanced Empirical Software Engineering* [32]. The principles are:

1. Setting the objectives
2. Survey design
3. Developing the survey instrument
4. Evaluating the survey instrument
5. Obtaining valid data

6. Analyzing the data

This chapter explains the objectives and practicalities i.e. how data is obtained, what tools and statistical methods for the analysis and what is the target group. The design and development and evaluation of instruments are explained in finer detail in the next chapter 6.

5.1 Objectives

Plenty of research has been done considering employees well-being and wellness at work but not particularly within the scope of software engineering nor especially combined with testing. One example of research on a similar topic is the study by Tsuneo, Yoshio and Kazuhiko presented on the previous chapter [30]. The objectives of this survey are to identify testing related habits and practices that can give an indication of better well-being at work.

Based on how well tests are written they can work as a good metric for describing the reliability of code. On the other hand, without a lack of sufficient knowledge or experience tests can also slow down the development for example by testing unnecessary sections or an attempt to reach 100% code coverage. It would be interesting to find a link between developers' expectations towards their code and their well-being. Could the response of whatever test metric be used, give a positive feeling and build reliability for the developer and therefore lowering their fear of publishing new features nor just being able rapidly to move on to the next requirement without worrying. The questions about reliability on the questionnaire attempt to find track kind of behaviour.

5.2 Target group

A target group for this survey is software developers who have experience working on a team or on a company. Work experience is used as one parameter to categorize samples.

The goal is to receive responses mostly from people who have worked in such projects where they have been part of developing software that has tests. The software can be built for in-house purposes or for a customer or even an open source project, but the interest is towards persons who have experience in testing.

Experience is measured by asking the respondents how many years they have worked on the software industry and what is their current work title. Other demographic information is gathered too such as size and type of the organization.

5.3 Data collection

A personal opinion survey is selected for the source of data due it can be distributed widely and using mainly closed questions simplifies the analysis of answers and enables to use of statistical methods more easily. The data for the survey is collected using an online survey tool Webropol [33]. The platform offers a simple interface for creating versatile questionnaires.

The questionnaire is shared across personal channels as well as email lists and networks using a social media platform LinkedIn. The results are stored on Webropol's servers in Finland only on the survey phase and the data is downloaded into a safe location right after the questionnaire closes.

The questionnaire can be found entirely on appendix A. It is divided into four sections. The first part focuses on well-being at work, stress and these questions are derived from the survey of Parker G. and Hyett M. [27]. The second part includes questions about a respondent company's software development and testing measures e.g. amount and type of tests. The existing study and survey on the industry practices were used as a base for this section [34]. The third part ask how the development feels and fault discovery. The last section is for collecting demographics.

An open text field is also added to the end of the questionnaire to collect possible

feedback and improvement ideas.

5.4 Analysis

Answers from the questionnaire are analysed using Webropol's built-in tools as well as statistical computing language R and Excel. On Webropol each question is given shorter but recognizable names e.g. converting the first question "Is your work fulfilling?" to "IsFulfilling". which makes handling and separating answers easier. All the questions besides demographics and testing tools are ranked using an ordinal Likert scale with options: "Not at all", "Slightly", "Moderately", "Very" and "Extremely true". The options are weighted from zero to four in the same order which then can be used performing further analysis on data. A multiple-choice field is used to collect data about different testing tools.

The questionnaire and its results can be downloaded from Webropol as on Excel format. The sheet contains each question on individual column and each row represents one respondent. Answers are mapped to integers from zero to four which respects Likert scale.

Webropol is used mainly for gathering basic information and getting an overview of the data. It offers a handy user interface for viewing each individual question, answer percentages, standard deviations and filtering. Excel's pivot table is used for the most analysis for comparing individual questions to each other and R is used to perform more advanced filtering and to plot a correlation heatmap. Results are gone through in the chapter 7.

6 Survey

This chapter introduces the topics and questions of the survey's questionnaire and discusses the categories and why each of them got selected and previous studies that worked as background sources. The survey will not aim to offer fully in-depth results for either work satisfaction nor about correct testing methodologies but instead to give an insight about possible relations between these factors and even to encourage for later studies.

There are no personal data or information collected for this survey which could be used to link answers to a single person. A voluntary email field which is used for contacting a winner of movie tickets is an only exception, but the information is separated from the survey results. The anonymous data could be shared with with other research interested on the topic.

The target group of this survey is software developers who have experience of working in a team or individually in such project where they have been a part of making new or existing software for in-house use, for a customer or a non-profit application. Understanding of testing tools and publication process is required from the respondents.

6.1 Cover letter structure

The structure of the questionnaire's cover letter follows the one presented on [32]. The cover letter can be found on appendix A.

1. A purpose of the survey

2. A description of the author
3. A cover letter
4. An explanation about the target group
5. An instructions to complete the questionnaire
6. An estimation of time to fill the questionnaire

6.2 Topics

The questionnaire consists of four topics: work well-being, software testing, software reliability and demographic information. Besides these topics, an optional extra field was added for the respondents to give feedback on the questionnaire and contact information. Contact information is used only for awarding one respondent with two movie tickets.

The first part *work well-being* maps respondents' personal experiences and feelings towards the workplace. The questions for the set are adapted from Gordon and Parker's *Measurement of Well-Being in the Workplace* (Black Dog Institute) survey in which they conducted a 31-item long online questionnaire about work well-being using four different factors: work satisfaction, organizational respect for the employee, employer care and intrusion of work into private life [27]. First and last factors were selected to be used in this questionnaire because they give a good enough indication of both well-being and stress.

In the second part *software testing* respondents are asked about testing tools, procedures and habits which are affecting or guiding the software development in their projects. Testing habits are collected by re-using questionnaire *Software Testing: Survey of the Industry Practices* [34]. Researchers created a study where they collected data from organizations and analyzed how organizations tested their products and what processes they followed.

The third *software reliability* part asks respondent's personal opinions about whether their development procedures i.e. publication pipelines, testing metrics, etc. are sufficient enough for them to create new releases which they can trust.

In the last part *background information* the respondents are asked about personal roles and experiences, details of their workplace i.e. the size of the company, size of the possible team, the industry and the nature together with the scale of the software which was developed.

6.3 Questions

6.3.1 Work well-being

The questionnaire's first two sets are concerning work satisfaction, well-being and stress. The first ten questions are used to measure for example how satisfactory work is. The second part contains seven questions and measures work-related stress and discomforting aspects of it.

6.3.2 Testing tools

Testing tools and practice questions were adapted from the existing survey study [34]. The survey's objective was to discover common industry practices used in software testing. They have performed the same questionnaire for two years, 2009 and 2017, and concluded that the industry has moved towards automated testing and organizations are applying more agile practices into their workflow than before. This gives a promising starting point for this survey to acknowledge that testing has become more popular. Five questions concerning software testing were selected from their study and are listed entirely in appendix A.

6.3.3 Software Reliability

The reliability of the software in this questionnaire was measured from a developer's perspective by asking experiences on how testing tools and data gained from them support development, making new features and releases. Reliability could be understood to construct from the number of defects, error distribution, code complexity, and design defect density [35]. Data is then derived from multiple questions. It would have been reasonable to add extra questions about quality models or services but that can be applied to later studies.

Questions "Please, estimate following claims concerning problems about testing":

1. Complicated testing tools cause test configuration errors.
2. It is difficult to automate testing because of low reuse and high price.
3. Insufficient communication slows the bug-fixing and causes misunderstanding between testers and developers.
4. Defining detailed test cases is inefficient due to large amount of revisions needed during the development work.
5. Feature development in the late phases of the product development shortens testing schedule.
6. Testing personnel do not have expertisein certain testing applications.
7. Our testing tools do not support our software process model.
8. Existing testing environments restrict testing.
9. Software maintenance costs are constantly growing.
10. We do not have the necessary tools to extract enough information for efficient maintenance.

6.3.4 Demographics

In order to be able to add more value to the study some background information about the respondent is requested. Demographic information is a valuable resource for performing thorough analysis which enables to inspect data from different perspectives and possibly to find interesting differences e.g. between experience levels or the size of the organization.

Questions:

1. Please, select option that best describes your current position
2. Please, select how many years you have worked on software industry?
3. Please, select the attributes, which describe your current organization

6.3.5 Feedback and contact information

A voluntary feedback field is added for respondents to write what they thought about the survey, collect improvement ideas and find flaws about the questions. Besides that, an email field is added to collect addresses for the lottery and that information will not be connected to the actual dataset. The tickets for one lucky respondent will be delivered after the results have been analysed and the thesis completed.

Questions:

1. Is there something else you would like to share or just leave a feedback about the survey?
2. Feel free to leave your email address if you want to participate in the raffle of movie tickets. Email is not used as part of the survey but for contacting the winner.

7 Results

7.1 Overview

The questionnaire (appendix A) was open for fifteen days in May of 2019 and during this time it gained 38 responses from people from different positions, organizations and with varying work experiences. The questionnaire was opened in total 241 times which can be seen from Webropol follow up data and this makes the response rate 15,7% which is a relatively good result for an internet survey [32]. On average the respondents spent 11.5 minutes on completing the survey.

Breakdown by demographics shows that most of the respondents (45%) were full-stack developers and the second most common roles were student and desktop or enterprise application developer both with the same 10 percentages (7.1). None of the respondents had worked for more than 15 years and the only one responder did not have any previous work experience on the software engineering when on the contrast more than half of the respondents had been working between one to five years (7.2). Almost all of the respondents work for private companies with 92 percentage which was expected (7.4). Different sized organizations were presented more evenly with the small (11 to 50 employees) being the largest covering 40 percent of answers. Unfortunately not a single freelancer or self-employed entrepreneur were reached (Table 7.3).

For an overview of the data a correlation analysis was performed which displays individual r-values of each question relative to another. Spearman's rank correlation is a

Position	n	Percent
Developer, full-stack	17	44.74%
Developer, desktop or enterprise applications	4	10.53%
Student	4	10.53%
Developer, embedded applications or devices	3	7.9%
DevOps specialist	3	7.89%
Developer, front-end	2	5.26%
Data or business analyst	1	2.63%
Designer	1	2.63%
Developer, game or graphics	1	2.63%
Developer, QA or test	1	2.63%
Product manager	1	2.63%

Table 7.1: Responses by position

recommended tool for comparing ordinal values therefore chosen to be used in this research [32].

Well-being

The indication of well-being was derived from the first two question sets. The first set focused on measuring well-being via work satisfaction i.e. how respondents felt about their current work situation and how well they fit into the job. The next part had seven questions which mapped respondent' stress, self-esteem and how much the work affects their private life.

In the study of Parker and Hyett, they derived well-being scores from each question set by summing up each response and computing an average and standard deviation. The questionnaire they conducted was published as an online survey which received over 1200 responses and the average and SD for the first set concerning well-being was 20,9 and 8,7 and for the fourth set considering stress the average and SD were 10,9 and 5,0 [27]. These

Work experience	n	Percent
No experience	1	2.63%
<1	8	21.05%
1-5	22	57.9%
6-10	6	15.79%
11-15	1	2.63%

Table 7.2: Responses by experience

Organization size	n	Percent
Not working in the software industry	2	5.26%
Very small (2-10 employees)	5	13.16%
Small (11-50 emp.)	15	39.47%
Medium (51-250 emp.)	5	13.16%
Large (250-1000 emp.)	8	21.05%
Very large (1001+ emp.)	3	7.9%

Table 7.3: Responses by organization size

Organization type	n	Percent
We are a government/municipal organization.	1	2.63%
We are a private company.	35	92.11%
We are a non-profit organization.	2	5.26%
We are open source developers.	3	7.89%
Our business is primarily national.	11	28.95%
Our business is primarily international.	9	23.68%
Our deliverables are mostly services.	12	31.58%
Our deliverables are mostly products.	13	34.21%

Table 7.4: Responses by organization type

same derived scores from our responses were 27,8 and 6,2 for the first and for the second set 4,6, and 5,2.

The average score of all well-being questions was 2.8 out of 4. Almost half of the respondents agreed that their current work offers enough challenges to advance on their skills but also that they have some level of independence at work. These results are slightly reflected negatively on the second set about stress and problems at work. The total average of the stress set was just 1.3 out of 4 which confirms the previous assumption about well-being. More than 86 percent responded, "Not at all" and "Slightly" to the question "Does your work impact negatively on your self-esteem?"

The well-being and stress correlations compared side by side reveals a couple of interesting aspects. The first thing to notice from a correlation heatmap (Fig. 7.1) of the first two sets is that questions about well-being and stress are opposite to each other but between them are relatively little or none statistically significant relationship. Highest negative correlation coefficient r -0.62 between the sets was "Is your work fulfilling?" and "Does your work impact negatively on your self-esteem?". This could indicate that people do want to perform well in their job but without being able to make advance or be able to enjoy what they are doing can indeed have effects on mental health. The small correlation can be explained with the assumption that the developers who answered to the questionnaire are feeling delighted about their job. Shortened questions are explained on appendix B.

Testing

The respondents were first asked about software development habits and ways how they share the knowledge within the organization. From the results it was clear that people prefer direct communication over written documentation and nearly 90% of responses of the question "Progress of the software is more important than thorough documentation" were "moderately" or more. All the questions of the set followed a similar pattern.

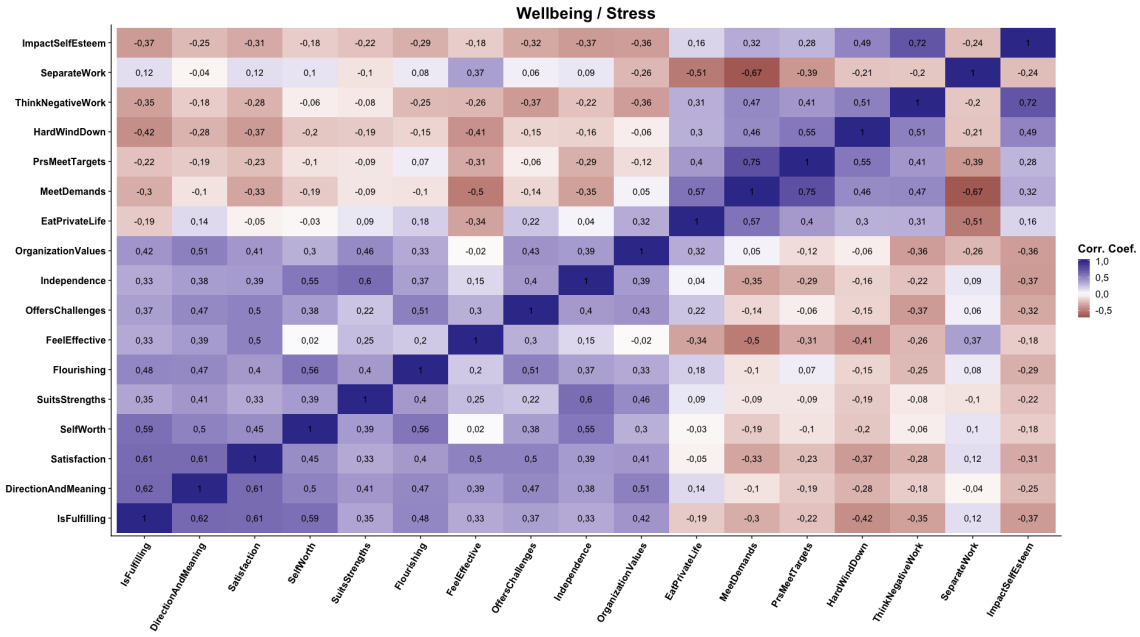

Figure 7.1: Correlation of well-being and stress answers per question

One respondent did not answer the questions about what faults the product can cause, otherwise 101 options were selected. The biggest problems were mere irritation and dissatisfaction, the second biggest problem was a disturbance in the normal operation of the organization, or a person and loss of human life/lives were selected by one respondent. The data collected from this set was not unfortunately used in this study as a metric due to lack of time but instead could be used in later studies.

Questions 5. (Table: 7.5) and 6. (Table: 7.6) were directly related to testing tools and how well they perform on the project. Overall respondents were agreeing (moderately or better) that their organization is building the product right but also that the product is right for its purpose. When asked what methods are used for validating the code quality and processes, code reviews arise above all with 37% of extremely true answers. Unit tests were the most used testing method (58%) in question 6. just above test automation and bug reporting (47% both). Curiously enough even though unit testing was most common it received a lower score than many others when asked about the quality of it in question 5.

Question	Not at all	Slightly	Moderately	Very	Extremely true
Our software correctly implements a specific function.					
We are building the product right.	0%	2.7%	32.43%	48.65%	16.22%
Our software is built traceable to customer requirements.					
We are building the right product.	2.7%	8.11%	18.92%	45.95%	24.32%
Our formal inspections are OK.	5.56%	11.11%	44.44%	30.56%	8.33%
We go through checklists.	2.63%	28.95%	36.84%	18.42%	13.16%
We keep code reviews.	13.16%	13.16%	18.42%	18.42%	36.84%
Our unit testing (modules or procedures) is excellent.	18.92%	24.32%	32.43%	18.92%	5.41%
Our integration testing (multiple components together) is excellent.	18.92%	21.62%	29.73%	21.62%	8.11%
Our usability testing (adapt software to users' work styles) is excellent.	13.89%	16.67%	33.33%	19.44%	16.67%
Our function testing (detect discrepancies between a program's functional specification and its actual behavior) is excellent.	16.22%	27.03%	24.32%	21.62%	10.81%
Our system testing (system meet requirements specification) is excellent.	8.11%	21.62%	45.95%	18.92%	5.4%
Our acceptance testing (users run the system in production) is excellent.	5.71%	22.86%	25.71%	31.43%	14.29%
We keep to our testing schedules.	8.11%	24.33%	32.43%	24.32%	10.81%
Last testing phases are kept regardless of the project deadline.	10.81%	21.62%	18.92%	40.54%	8.11%
We allocate enough testing time.	5.41%	29.73%	35.13%	18.92%	10.81%

Table 7.5: Question 5: Please, estimate following claims concerning your software testing.

Question	n	Percent
Test case management	10	26.32%
Unit testing	22	57.89%
Integration testing	14	36.84%
System testing	16	42.11%
Test automation	18	47.37%
Performance testing	9	23.68%
Security testing	5	13.16%
Bug reporting	18	47.37%
Test design	6	15.79%
Quality control	12	31.58%
Bug/Code tracing	16	42.11%
Test completeness	13	34.21%
Automated metrics collector	10	26.32%
Virtual test environment	16	42.11%
Protocol/Interface conformance tool	1	2.63%
Other important tool: Please specify	2	5.26%
None	9	23.68%

Table 7.6: Question 6: Please select all of the applicable categories. Our organization has a dedicated tool, which manages the following testing aspect.

Question	Not at all	Slightly	Moderately	Very	Extremely true
Complicated testing tools cause test configuration errors.	21.21%	42.43%	21.21%	12.12%	3.03%
It is difficult to automate testing because of low reuse and high price.	26.47%	26.47%	29.41%	17.65%	0%
Insufficient communication slows the bug-fixing and causes misunderstanding between testers and developers.	23.53%	11.77%	23.53%	29.41%	11.76%
Defining detailed test cases is inefficient due to large amount of revisions needed during the development work.	11.11%	33.33%	25%	22.22%	8.34%
Feature development in the late phases of the product development shortens testing schedule.	20.59%	23.53%	32.35%	11.77%	11.76%
Testing personnel do not have expertise in certain testing applications.	34.37%	28.13%	25%	6.25%	6.25%
Our testing tools do not support our software process model.	60%	23.34%	13.33%	3.33%	0%
Existing testing environments restrict testing.	59.37%	6.25%	18.75%	6.25%	9.38%
Software maintenance costs are constantly growing.	20.59%	50%	17.65%	8.82%	2.94%
We do not have the necessary tools to extract enough information for efficient maintenance.	32.26%	41.94%	19.35%	6.45%	0%

Table 7.7: Question 7: Please, estimate following claims concerning problems about testing.

On average respondents have only slight problems with their testing tools. Answers of the questions 7. (Table 7.7) shows that the biggest problem is insufficient communication which slows down the development.

Software reliability

Question 8. (Table 7.8) aimed to find how code defects are noticed compared to how easy the process of development is. More than 60% responded "very" or "extremely true" when asked how pleasant the development is similar to adding new features and making new releases is rather easy for most.

7.2 Comparison

As stated before respondents were mostly pleased with their current situation and the same can be confirmed from a Figure 7.2 which displays how answers are scattered between how fulfilling work is and how pleasant the current project is to develop. Bigger value is better in both questions.

Question	Not at all	Slightly	Moderately	Very	Extremely true
Developing current software project is pleasent	0%	15.79%	21.05%	42.11%	21.05%
Adding new features is effortless process	5.26%	10.53%	44.74%	31.58%	7.89%
Releasing a new version into production is effortless process	5.26%	15.79%	31.58%	31.58%	15.79%
Tests are the main source of discovering faults and errors	13.16%	15.79%	42.11%	23.68%	5.26%
Customer or software users reports errors	5.26%	15.79%	28.95%	39.47%	10.53%
Urgent faults or errors discovered after a release	18.42%	36.84%	31.58%	7.9%	5.26%

Table 7.8: Question 8: Please, estimate following claims concerning your current (or latest) software project.

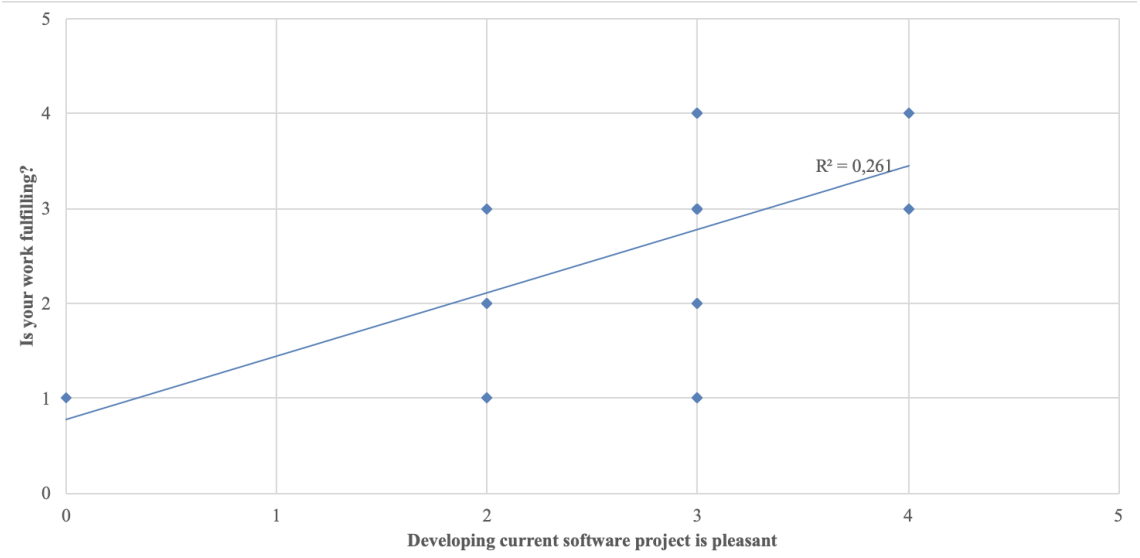

Figure 7.2: Result how fulfilling developing current project is.

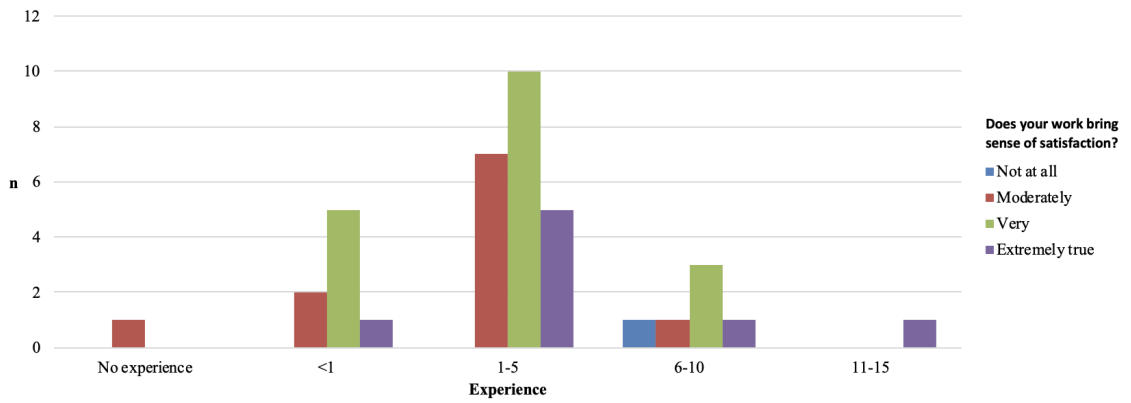

Figure 7.3: Satisfaction and experience.

Figure 7.3 displays how answers are distributed over different experiences. The middle option "1-5" years of work experience was the most common option and curiously a similar pattern can be seen around other options too. Satisfaction is spread pretty evenly regardless of the experience level but still it does not prove much due to small sample size.

Other popular development and testing method besides unit testing was a code review with 33 "slightly" or more answers. Figure 7.4 shows how respondents measure work being more fulfilling when they perform code reviews. An almost identical pattern can be seen when comparing to unit tests but it was left out due to its similarity.

Figure 7.5 presents average counts of testing methods grouped by sense of satisfaction. Counts were calculated by summarizing all selected testing tools by rows (question 9. Appendix A) and computing averages. The total average of the responses was five tools at use. The figure indicates that more testing tools might not be better but instead respondents with a better feeling of satisfaction have also selected fewer tools.

Separation of work has a high negative correlation between "Defining detailed test cases is inefficient due to a large number of revisions needed during the development work" and "Feature development in the late phases of the product development shortens testing schedule" and similar trend is noticeable if testing tools do not support the software process model.

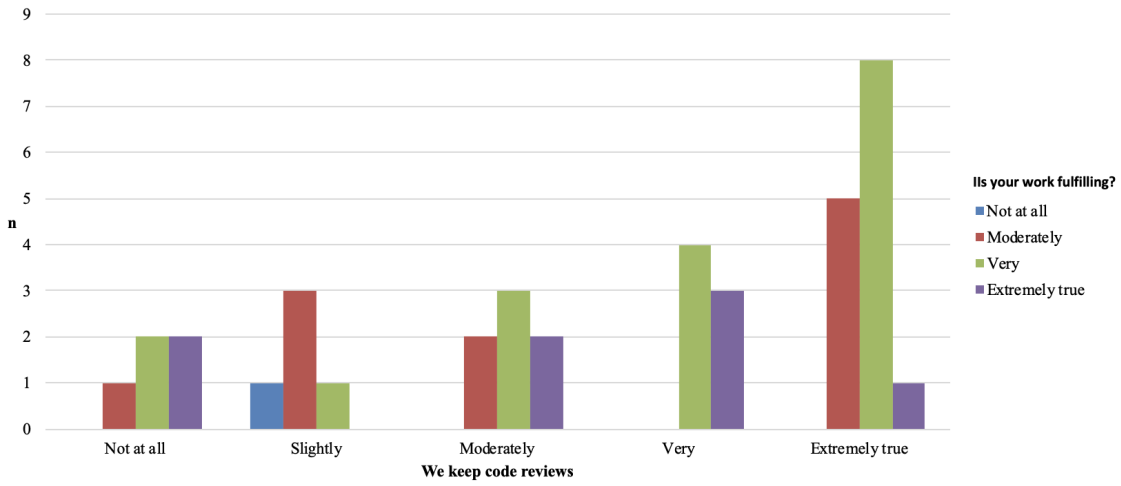

Figure 7.4: Code reviews and work fulfillness.

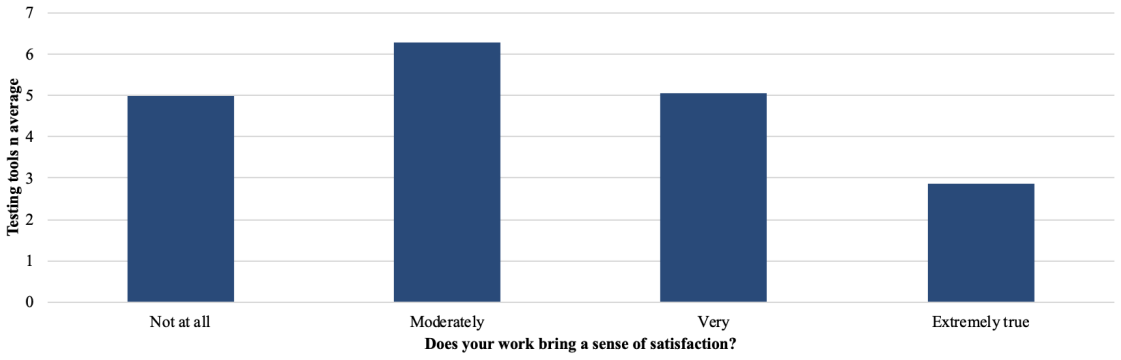

Figure 7.5: Amount of testing tools and sense of satisfaction.

7.3 Discussion

Responses were received from a rather good variety of developers from different types of organizations and experience levels which gave a proper base for the study.

Generally, testing seems to be a common and used practice in the software industry nowadays. Unit testing and test automation were the most used tools which are the same result that Hynninen concluded in their industry practices study [34].

Results considering well-being showed promising information about developers' attitudes towards their work. Our results are in line with the results from both Parker and Hyett's, and StackOverflow's studies. The average scores correlated directly with Parker and Hyett's results and a similar positive attitude towards work and tools can be seen here as what StackOverflow concluded. [27]

Based on these findings it can be said that there is a connection between testing tools and employee's well-being. Even though the reliability derived from adding new features and pleasure of development shows a correlation with fulfillment these claims still require more studying. The dataset is large and contains many questions that were not part of the analysis this time. The credibility of the study could be increased by gathering more responses and possibly leave out some questions to make the questionnaire a bit easier and effortless to fill.

8 Conclusion

This thesis explores the possible connection between well-being and software development and attempts to find the answer whether work well-being could be improved solely by proper testing. These topics are studied and presented individually in the early chapters and that knowledge is used to create a questionnaire that strived to find a correlation between software developers' well-being and testing habits.

The first chapter is an introduction to software development processes and compares how former models like the waterfall model differ from the ones used today. There is no one clear and perfect model but instead, during the last circa twenty years the agile software development has taken over the industry as it has displaced many of the old methods. It presented a whole new ideology with *Agile manifesto* which then has been used as a base for many frameworks like the project management framework Scrum and Extreme Programming.

More importantly the agile movement has introduced plenty of development concepts like Test-driven development (TDD) which are easy for individual developers to follow and guide towards creating reliable and robust software. The main focus of the second chapter is on testing methodologies starting from TDD but also various other tools are discussed that are in a key role in the survey.

Software and development processes have been noticed to be as important assets as any other industrial product or manufacturing process and due to this many international standards have been formed to control and improve the software quality. The beginning of

chapter three described different standards and how the quality of software is measured.

Chapter four introduced different definitions and estimation models for well-being but the main focus was on work well-being and workplace wellness. Stress and mental illnesses have been noticed to have an effect on productivity and this phenomenon was used as a key metric in this study.

For the practical part of the thesis an online questionnaire was conducted, and it received a good amount of 38 responses from software engineers with different backgrounds and experiences. The structure, methods and results of the questionnaire were explained in chapters five to seven. The questionnaire was split into four sections: well-being, testing, reliability and demographics where the first two sections were derived from two existing instruments. The instruments included a survey base for well-being and testing both of which have been tested in practice and were intended to be used in further research.

The data shows that the majority of respondents were very happy with themselves and the tools they are using at their organization. On average well-being was clearly experienced high whereas similarly stress was experienced low. The results also indicate that in some cases descent amount of testing tools do increase well-being or at least a sense of satisfaction and feeling of fulfilment in the organization. Reliability towards the software was assessed by asking about the meaningfulness of developing the software and these results also showed promising but not concrete results towards the hypothesis.

Unfortunately, strict statistical evidence cannot be concluded from the data mostly because of a small sample size but still, the results give a strong indication towards that there indeed is a connection between well-being, software testing and reliability. The response dataset also contains lots of questions that were not used to derive these results so even the data itself offers a lot more possibilities to study. The same questionnaire could be used as it is or as a base for later studies for example to find concrete problems which prevent developers to work at their best.

References

[1] P. Y. Chen and C. L. Cooper, *Wellbeing: a Complete Reference Guide, Work and Wellbeing. Volume III.* Wiley-Blackwell [Imprint], 2014, ISBN: 9781118716212.

[2] Standish Group, "Chaos Report 2015", Tech. Rep., 2015, p. 13. [Online]. Available: https://www.standishgroup.com/sample%7B%5C_%7Dresearch%7B%5C_%7Dfiles/CHAOSReport2015-Final.pdf.

[3] H. Mohanty, J. Mohanty, and A. Balakrishnan, *Trends in Software Testing*, ISBN: 9789811014147. DOI: 10.1007/978-981-10-1415-4. [Online]. Available: https://link-springer-com.ezproxy.utu.fi/content/pdf/10.1007%7B%5C%%7D2F978-981-10-1415-4.pdf.

[4] S. I. Gatchel R.J., *Handbook of occupational health and wellness, handbooks in health, work and disability*, ser. Handbooks in Health, Work, and Disability. Springer Science + Business Media, 2012, p. 572, ISBN: 1461448395.

[5] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, Inc, 2009, ISBN: 9780132350884. DOI: 0132350882. arXiv: arXiv:1011.1669v3.

[6] G. O'Regan, "Concise Guide to Software Engineering", in *Concise Guide to Software Engineering*, Cork, Ireland: Springer, 2017, pp. 131–138, ISBN: 978-3-319-57750-0. DOI: 10.1007/978-3-319-57750-0. [Online]. Available: http://www.springer.com/series/7592%20http://link.springer.com/10.1007/978-3-319-57750-0.

[7] W. W. Rovce, "MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS", Tech. Rep., 1970, p. 11. [Online]. Available: http://www-scf.usc.edu/%7B~%7Dcsci201/lectures/Lecture11/royce1970.pdf.

[8] Agilealliance, *What is Agile Software Development? | Agile Alliance*. [Online]. Available: https://www.agilealliance.org/agile101/ (visited on 03/17/2019).

[9] *The art of software testing*. John Wiley & Sons, 2012, p. 240, ISBN: 9781118133132.

[10] R. O. Rogers, "Acceptance Testing vs. Unit Testing: A Developer's Perspective", in *Extreme Programming and Agile Methods - XP/Agile Universe 2004*, Springer, Berlin, Heidelberg, 2004, pp. 22–31. DOI: 10.1007/978-3-540-27777-4_3. [Online]. Available: http://link.springer.com/10.1007/978-3-540-27777-4%7B%5C_%7D3.

[11] H. Naik, "Behavior Driven Development: An Effective Technical Practice to Develop Good Software", *International Journal of Computer Applications*, vol. 149, no. 5, pp. 23–27, 2016. DOI: 10.5120/ijca2016911400. [Online]. Available: https://www.ijcaonline.org/archives/volume149/number5/naik-2016-ijca-911400.pdf.

[12] M. Fowler, *The Practical Test Pyramid*. [Online]. Available: https://martinfowler.com/articles/practical-test-pyramid.html (visited on 03/26/2019).

[13] C. Climate, *The Rails Testing Pyramid*, 2013. (visited on 06/09/2019).

[14] A. M. J. Hass, *Guide to advanced software testing*. Artech House, 2008, p. 427, ISBN: 1596932856.

[15] G. Adzic, *Specifcation by Example*. Manning, 2008, p. 295. [Online]. Available: https://the-eye.eu/public/Books/IT%20Various/specification%7B%5C_%7Dby%7B%5C_%7Dexample.pdf.

[16] ISO, *ISO/IEC/IEEE 90003:2018 Software engineering - Guidelines for the application of ISO 9001:2015 to computer software*, 2018. DOI: 10.1109/IEEESTD.2018.8559961. [Online]. Available: https://utu.finna.fi/PrimoRecord/pci.ieee10.1109%7B%5C%%7D2FIEEESTD.2018.8559961%20https://www.iso.org/standard/74348.html.

[17] IEEEE1990, *610.12-1990 IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1990, ISBN: 0738103918. [Online]. Available: https://utu.finna.fi/Record/nelli30.3780000000093000.

[18] D. Galin, "Software Quality Assurance From theory to implementation", Tech. Rep., 2004, p. 617. [Online]. Available: www.pearsoned.co.uk.

[19] J. McCall, P. Richards, and G. Walters, "Factors in Software Quality", p. 168, 1977.

[20] K. Naik and P. Tripathy, "Software Testing and Quality Assurance : Theory and Practice", Tech. Rep., 2008. [Online]. Available: http://ebooks.bharathuniv.ac.in/gdlc1/gdlc1/Software%20Engineering/SOFTWARE%20TESTING%20AND%20QUALITY%20ASSURANCE,Theory%20and%20Practice%20-%20KSHIRASAGAR%20NAIK.pdf.

[21] B. Singh and S. P. Kannojia, "A Review on Software Quality Models", in *2013 International Conference on Communication Systems and Network Technologies*, IEEE, Apr. 2013, pp. 801–806, ISBN: 978-1-4673-5603-9. DOI: 10.1109/CSNT.2013.171. [Online]. Available: http://ieeexplore.ieee.org/document/6524514/.

[22] ISO, *About us*. [Online]. Available: https://www.iso.org/about-us.html (visited on 04/01/2019).

[23] Iso, "ISO 9000 Quality management systems - Fundamentals and vocabulary", Tech. Rep., 2015. [Online]. Available: https://www.iso.org/standard/45481.html.

[24] P. Rosati, P. Deeney, M. Cummins, L. Van Der Werff, and T. Lynn, "Social media and stock price reaction to data breach announcements_ Evidence from US listed companies", 2018. DOI: 10.1016/j.ribaf.2018.09.007. [Online]. Available: https://doi.org/10.1016/j.ribaf.2018.09.007.

[25] B. Marick and T. Foundations, "How to Misuse Code Coverage", Tech. Rep., 1997. [Online]. Available: http://www.exampler.com/testing-com/writings/coverage.pdf.

[26] N. O'Reilly, "Wellbeing: The five essential elements", *The Journal of Positive Psychology*, vol. 8, no. 2, pp. 174–176, Mar. 2013, ISSN: 1743-9760. DOI: 10.1080/17439760.2013.765502. [Online]. Available: https://doi.org/10.1080/17439760.2013.765502.

[27] G. B. Parker and M. P. Hyett, "Measurement of Well-Being in the Workplace", *The Journal of Nervous and Mental Disease*, vol. 199, no. 6, pp. 394–397, Jun. 2011. DOI: 10.1097/NMD.0b013e31821cd3b9. [Online]. Available: https://insights.ovid.com/crossref?an=00005053-201106000-00007.

[28] World Health Organization, "Healthy workplaces: a WHO global model for action", Tech. Rep., 2010, pp. 1–32. [Online]. Available: https://www.who.int/occupational%7B%5C_%7Dhealth/publications/healthy%7B%5C_%7Dworkplaces%7B%5C_%7Dmodel%7B%5C_%7Daction.pdf%20https://www.who.int/occupational%7B%5C_%7Dhealth/healthy%7B%5C_%7Dworkplaces/en/.

[29] Nash and L. Nash, *Success That lasts*, 2. Graduate School of Business Administration, Harvard University, 2004, vol. 82, pp. 102–109. [Online]. Available: https://utu.finna.fi/PrimoRecord/pci.proquest227768061.

[30] T. Furuyama, Y. Arai, and K. Iio, "Analysis of fault generation caused by stress during software development", *Journal of Systems and Software*, vol. 38, no. 1, pp. 13–25, Jul. 1997, ISSN: 01641212. DOI: 10.1016/S0164-1212(97)00064-2. [Online]. Available: https://www-sciencedirect-com.ezproxy.utu.fi/science/article/pii/S0164121297000642.

[31] Stack Overflow, "Stack Overflow Developer Survey 2019", Tech. Rep., 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019?utm%7B%5C_%7Dsource=so-owned%7B%5C&%7Dutm%7B%5C_%7Dmedium=blog%7B%5C&%7Dutm%7B%5C_%7Dcampaign=dev-survey-2019%7B%5C&%7Dutm%7B%5C_%7Dcontent=launch-blog%7B%5C#%7Dwork-%7B%5C_%7D-how-do-developers-feel-about-their-careers-and-jobs.

[32] F. Shull, J. Singer, and D. I. Sjøberg, *Guide to advanced empirical software engineering*. 2008, ISBN: 9781848000438. DOI: 10.1007/978-1-84800-044-5.

[33] *VIEW by Webropol - Johda tiedolla | Webropol*. [Online]. Available: https://webropol.fi/ (visited on 04/16/2019).

[34] T. Hynninen, J. Kasurinen, A. Knutas, and O. Taipale, "Software testing: Survey of the industry practices", in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, IEEE, May 2018, pp. 1449–1454, ISBN: 978-953-233-095-3. DOI: 10.23919/MIPRO.2018.8400261. [Online]. Available: https://ieeexplore.ieee.org/document/8400261/.

[35] M. Li and C. S. Smidts, "A ranking of software engineering measures based on expert opinion", eng, *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 811–824, 2003, ISSN: 0098-5589. [Online]. Available: https://ieeexplore.ieee.org/document/1232286.

Appendix A Questionnaire

The questionnaire used in this thesis can be found entirely from below.

Relationship Between Software Testing, Software Quality and Work Well-being

Work well-being can affect either positively or negatively to a person's life hence studies around it has gained more popularity. Similarly, software development can be hectic and battling from a deadline to another. By investing in better software practices across the organization or teams can lighten the workload and therefore possibly even increase well-being.

The purpose of this study is to examine possible relationships between software testing habits of software companies and work well-being i.e. answer the question: can good testing practices and tools relieve stress and simplify obnoxious tasks and thereby improve developer's well-being.

It will take 10 to 15 minutes to complete the survey. The questionnaire consists of 10 sections covering questions about work well-being, testing tools, software quality assurance, and personal experience.

The answers to the questionnaire will be compiled in a report part of my master's thesis, which will be published in summer 2019. Movie tickets will be drawn among the participants. If you want to participate in the lottery, please provide your email at the end of the questionnaire. This information is not used in the survey or shared with any third parties.

If you have any questions regarding the survey, please don't hesitate to contact me at sakrnie@utu.fi.

Thank you for your contribution,
Sami Nieminen
MSci Student of Information and Communication Technology at the University of Turku

Instruments used making the questionnaire:
- Parker, B. (2011). Measurement of Well-Being in the Workplace: The Development of the Work Well-Being Questionnaire. The Journal of Nervous and Mental Disease, 199(6), pp. 394-397. doi:10.1097/NMD.0b013e31821cd3b9
- Hynninen, T. (2018). Software testing: Survey of the industry practices

Work Satisfaction

1. Please, estimate how the following claims describe your feelings towards your current work *

	Not at all	Slightly	Moderately	Very	Extremely true
Is your work fulfilling?	○	○	○	○	○
Do your daily work activities give you a sense of direction and meaning?	○	○	○	○	○
Does your work bring a sense of satisfaction?	○	○	○	○	○
Does your work increase your sense of self-worth?	○	○	○	○	○
Does your job allow you to recraft your job to suit your strengths?	○	○	○	○	○
Does your work make you feel that, as a person, you are flourishing?	○	○	○	○	○
Do you feel capable and effective in your work on a day-to-day basis?	○	○	○	○	○
Does your work offer challenges to advance your skills?	○	○	○	○	○
Do you feel you have some level of independence at work?	○	○	○	○	○
Do you feel personally connected to your organization’s values?	○	○	○	○	○

Intrusion of Work Into Private Life

2. Please, estimate how the following claims describe your experiences *

	Not at all	Slightly	Moderately	Very	Extremely true
Does your work eat into your private life?	○	○	○	○	○
Do you feel stressed in organizing your work time to meet demands?	○	○	○	○	○
Do you feel excessively pressured at work to meet targets?	○	○	○	○	○
After work, do you find it hard to wind down?	○	○	○	○	○
Do you find yourself thinking negatively about work outside of work hours?	○	○	○	○	○
Do you feel that you can separate yourself easily from your work when you leave for the day?	○	○	○	○	○
Does your work impact negatively on your self-esteem?	○	○	○	○	○

Software Quality Assurance Processes

3. Please, estimate how the following claims describe software development on your company or in your team?

	Not at all	Slightly	Moderately	Very	Extremely true
We like to transfer knowledge more by face-to-face conversation than by documents as the primary method of knowledge transfer.	○	○	○	○	○
Progress of the software is more important than thorough documentation.	○	○	○	○	○
Business people and developers work daily together in the projects.	○	○	○	○	○
Our process is able to cope with late changes in requirements, design, and technical platform.	○	○	○	○	○
We prefer more individuals, collaboration, and interaction than processes and tools.	○	○	○	○	○

Software Quality Assurance Processes

4. Faults in your products can cause (please, select all suitable points)

- [ ] Irritation and dissatisfaction
- [ ] Disturbance in the normal operation of the organization or a person
- [ ] Remarkable economical losses
- [ ] Interruption in the normal operation of the organization or a person
- [ ] Loss of human life/lives

Software Quality Assurance Processes

5. Please, estimate following claims concerning your software testing.

When the claim is not applicable leave the scale empty.

	Not at all	Slightly	Moderately	Very	Extremely true
Our software correctly implements a specific function. We are building the product right.	○	○	○	○	○
Our software is built traceable to customer requirements. We are building the right product.	○	○	○	○	○
Our formal inspections are OK.	○	○	○	○	○
We go through checklists.	○	○	○	○	○
We keep code reviews.	○	○	○	○	○
Our unit testing (modules or procedures) is excellent.	○	○	○	○	○
Our integration testing (multiple components together) is excellent.	○	○	○	○	○
Our usability testing (adapt software to users' work styles) is excellent.	○	○	○	○	○
Our function testing (detect discrepancies between a program's functional specification and its actual behavior) is excellent.	○	○	○	○	○
Our system testing (system meet requirements specification) is excellent.	○	○	○	○	○
Our acceptance testing (users run the system in production) is excellent.	○	○	○	○	○
We keep to our testing schedules.	○	○	○	○	○
Last testing phases are kept regardless of the project deadline.	○	○	○	○	○
We allocate enough testing time.	○	○	○	○	○

Testing and Quality Assurance Tools

6. Please select all of the applicable categories. Our organization has a dedicated tool, which manages the following testing aspect *

- [ ] Test case management
- [ ] Unit testing
- [ ] Integration testing
- [ ] System testing
- [ ] Test automation
- [ ] Performance testing
- [ ] Security testing
- [ ] Bug reporting
- [ ] Test design
- [ ] Quality control
- [ ] Bug/Code tracing
- [ ] Test completeness
- [ ] Automated metrics collector
- [ ] Virtual test environment
- [ ] Protocol/Interface conformance tool
- [ ] Other important tool: Please specify

_____

- [ ] None

Testing and Quality Assurance Tools

7. Please, estimate following claims concerning problems about testing.

When the claim is not applicable leave the scale empty.

	Not at all	Slightly	Moderately	Very	Extremely true
Complicated testing tools cause test configuration errors.	○	○	○	○	○
It is difficult to automate testing because of low reuse and high price.	○	○	○	○	○
Insufficient communication slows the bug-fixing and causes misunderstanding between testers and developers.	○	○	○	○	○
Defining detailed test cases is inefficient due to large amount of revisions needed during the development work.	○	○	○	○	○
Feature development in the late phases of the product development shortens testing schedule.	○	○	○	○	○
Testing personnel do not have expertise in certain testing applications.	○	○	○	○	○
Our testing tools do not support our software process model.	○	○	○	○	○
Existing testing environments restrict testing.	○	○	○	○	○
Software maintenance costs are constantly growing.	○	○	○	○	○
We do not have the necessary tools to extract enough information for efficient maintenance.	○	○	○	○	○

Software reliability

8. Please, estimate following claims concerning your current (or latest) software project *

	Not at all	Slightly	Moderately	Very	Extremely true
Developing current software project is pleasent	○	○	○	○	○
Adding new features is effortless process	○	○	○	○	○
Releasing a new version into production is effortless process	○	○	○	○	○
Tests are the main source of discovering faults and errors	○	○	○	○	○
Customer or software users reports errors	○	○	○	○	○
Urgent faults or errors discovered after a release	○	○	○	○	○

Work Experience

9. Please, select option that best describes your current position. *

- ◯ Academic researcher
- ◯ Data or business analyst
- ◯ Data scientist or machine learning specialist
- ◯ Database administrator
- ◯ Designer
- ◯ Developer, back-end
- ◯ Developer, desktop or enterprise applications
- ◯ Developer, embedded applications or devices
- ◯ Developer, front-end
- ◯ Developer, full-stack
- ◯ Developer, game or graphics
- ◯ Developer, mobile
- ◯ Developer, QA or test
- ◯ DevOps specialist
- ◯ Educator
- ◯ Engineer, data
- ◯ Engineer, site reliability
- ◯ Engineering manager
- ◯ Job title
- ◯ Product manager
- ◯ Scientist
- ◯ Student
- ◯ System administrator

10. Please, select how many years you have worked on software industry? *

◯	◯	◯	◯	◯	◯
No experience	<1	1-5	6-10	11-15	>15

11. What is the size of your current organization *

- ◯ Not working in the software industry
- ◯ Self employed/freelancer
- ◯ Very small (2-10 employees)
- ◯ Small (11-50 emp.)
- ◯ Medium (51-250 emp.)
- ◯ Large (250-1000 emp.)
- ◯ Very large (1001+ emp.)

12. Please, select the attributes, which describe your current organization *

- ☐ We are a government/municipal organization.
- ☐ We are a private company.
- ☐ We are a non-profit organization.
- ☐ We are open source developers.
- ☐ Our business is primarily national.
- ☐ Our business is primarily international.
- ☐ Our deliverables are mostly services.
- ☐ Our deliverables are mostly products.

13. Is there something else you would like to share or just leave a feedback about the survey?

_____

_____

_____

_____

_____

14. Feel free to leave your email address if you want to participate in the raffle of movie tickets. Email is not used as part of the survey but for contacting the winner.

Email

Appendix B Questionnaire schema

	Column	Question	Set
1	IsFulfilling	Is your work fulfilling?	Please, estimate how the following claims describe your feelings towards your current work
2	DirectionAndMeaning	Do your daily work activities give you a sense of direction and meaning?	
3	Satisfaction	Does your work bring a sense of satisfaction?	
4	SelfWorth	Does your work increase your sense of self-worth?	
5	SuitsStrengths	Does your job allow you to recraft your job to suit your strengths?	
6	Flourishing	Does your work make you feel that, as a person, you are flourishing?	
7	FeelEffective	Do you feel capable and effective in your work on a day-to-day basis?	
8	OffersChallenges	Does your work offer challenges to advance your skills?	
9	Independence	Do you feel you have some level of independence at work?	
10	OrganizationValues	Do you feel personally connected to your organization's values?	
11	EatPrivateLife	Does your work eat into your private life?	Please, estimate how the following claims describe your experiences
12	MeetDemands	Do you feel stressed in organizing your work time to meet demands?	
13	PrsMeetTargets	Do you feel excessively pressured at work to meet targets?	
14	HardWindDown	After work, do you find it hard to wind down?	
15	ThinkNegativeWork	Do you find yourself thinking negatively about work outside of work hours?	
16	SeparateWork	Do you feel that you can separate yourself easily from your work when you leave for the day?	
17	ImpactSelfEsteem	Does your work impact negatively on your self-esteem?	

Table B.1: Questionnaire well-being and stress schema