The Human Factors and the Hard Skills Shaping the Role of the Software Tester in Agile Teams

A Grounded Theory Study

Lucas Paruch

Thesis submitted for the degree of
Master in Informatics: Programming and System Architecture
60 credits

Institute for Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2020

The Human Factors and the Hard Skills Shaping the Role of the Software Tester in Agile Teams

A Grounded Theory Study

Lucas Paruch

The Human Factors and the Hard Skills Shaping the Role of the Software Tester in Agile Teams

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

My Master Thesis aims to fill a research gap within the area of software testing, focusing on the human dimension of the software testing role. Although there has been extensive research on the technological aspect of software testing - such as tool usage, test automation, and test processes - little research has been conducted within the human factors shaping the role of the software testers.

In the first part of my thesis, I demonstrate this gap, by conducting two systematic literature reviews. I continue by presenting the design, and implementation of a qualitative analysis, and by creating a theory on the combination of human factors and hard skills that shape the software testing role, as described by the professionals working with software.

The qualitative study was conducted in collaboration with an international software service providing company, who facilitated the interviewing of different professionals, as well as allowing me to observe a team developing a customer-project within the financial sector. I interviewed a total of 13 professionals, six with software testing roles and seven with non-testing roles, and observed over 14 workdays on the project.

The research method applied was Grounded Theory along with Thematic Analysis, well suited given the scope of the thesis, as well as the data collection and analysis. Through interviews and observations, I obtained a set of human factors and hard skills as described by the interviewed professionals, and sequentially, I obtained an emerging theory on how these factors influence one another.

The qualitative analysis findings suggest that the professionals working with agile software development see the following qualities as definitory for the software testing role: being adaptable, detail-oriented, creative, structured, curious, having good communication skills, and able to see the whole picture. The emerged theory shows that for software testers, both domain knowledge and technical knowledge will have a influence on being creative, adaptable, detail-oriented, and being able to see the whole picture. In addition, being structured will positively impact the ability to be detail-oriented. Software testers that are constantly curious, are able to learn domain-knowledge and technical-knowledge at an easier and faster pace. The theory also shows that having good communication skills, being detail-oriented, and able to see the whole picture as a software tester will benefit the team's effectivity.

This thesis' findings contribute by bringing to light the knowledge about the human factors and their combination that give uniqueness to the software testing role. For practitioners, this thesis can benefit the industry by providing factors that testers should possess, and useful as a check-list to those in charge of hiring new testing personnel. For researchers, this thesis contributes by adding knowledge to the little research available in this field, and can serve as a foundation for future work.

Acknowledgements

Writing this Master's Thesis has been a unique experience in a challenging yet rewarding way. Firstly, I would like to give commendation to my supervisor, Viktoria Stray. I am deeply grateful for her tremendous support and insight. Her valuable knowledge, experience, feedback, as well as our lively discussions paved the way to different ideas, many of which are currently implemented in the thesis. I also wish to extend my gratitude to Raluca Florea for her incredible knowledge in the software testing field, her exceptional input and guidance helped me shape the thesis into its current state.

I would like to express my thankfulness to Charlotte Bech Blindheim. The company collaboration set-up would not have been possible without her encouragement and continuous support. A special thanks goes to the interview participants, for allowing me to conduct interviews, and to collect data, as well as to the software team members, for allowing me to conduct observations on project Sierra.

I thank my fellow students and the employees of the research group Programming and Software Engineering for fruitful discussions and help. Lastly, I owe deep gratitude to my family for their continuous support and inspiration - for I would not be where I am today without your support.

PS: if there are any errors in this thesis - as surely there must be - they are probably the fault of Viktoria as she is my supervisor and really should have trained me better :)

Preface

In 2015, I enrolled into the study program called Informatics: Programming and Network. During the first two years, I learned basic programming skills, as well as got used to think as a developer. Two years later, I chose to enroll in a course that was relatively new at the time - Software Testing. Thus, the doors into the world of testing opened up for me. During the course, I learned different testing types, techniques to test the software, and how to think like a tester. I hadn't done anything like this before, but the more I learned, the more intrigued I became. It was a unique and educational experience, that made me become a Teacher's Assistant twice for the course, during my time at the university. However, I longed for more practical experience, as the most of the course was theoretical. How was testing actually performed in the real world? An opportunity presented itself when a software service providing company offered me a test-coordinator summer internship.

I began working at the company as an intern during the summer of 2019. We, the students benefiting of this apprenticeship, were placed into cross-functional teams and asked to be working in actual projects. The company's software development methodology was agile, therefore we faced a multitude of tasks and changes, in a relatively short time span. Most of the curriculum I learned at the testing course were not applicable in this kind of projects; needless to say, I felt out of place. I gained a consistent amount of practical experience during that summer, for instance on how testers are the 'middle-layer' between the technical and business side. I assisted the UX professionals in shaping the user-experience and provided them with technical insights from the developers' stand point. I helped the developers in understanding the software architecture while translating the UX requirements into more concrete technical details. From the business side, I performed risk assessment together with the business analyst and acquired good domain knowledge - which was furthermore translated into technical detail. I still had my own testing tasks, such as creating test plans, test reports, making test-cases, and reporting bugs.

After my internship, I realized that my perception of testers was inaccurate; they did much more than test the code produced by the developers. Testers needed to possess good communication and interpersonal skills, being able to work well in a team, regularly adapt themselves, and needed to possess a sense of commitment. The book which we used as the curriculum in the course [8], based on the ISTQB Foundations syllabus - had little mention within this regard. Coming back from the internship and onto my last year of Master's, I realised this topic was something I wanted to research. The motivation for writing this Master's Thesis therefore originated from my personal experience as a software testing course participant, and as an intern. With anticipation, this thesis would hopefully benefit both the research field, as well as the industry focusing on software testing.

Contents

List of Tables

List of Figures

1 Introduction

Agile software development began in the late 1990s, as organizations found it burdensome to apply sequential approaches for small to medium-sized projects [80]. Its methods are based on iterative and incremental development, characterized by qualities such as short time-boxed or task-boxed cycles, and rapid customer feedback. Today, methodologies such as Scrum, are prevalent among agile organizations. The 12th annual report from CollabNet VersionOne - the most significant and longest-running agile survey in the world - revealed that over 50% of the 1400 respondents' organizations are currently practicing Scrum [18]. From the perspective of software testing, agile might appear radical at first. The short iterations might give the impression that agile does not allow as much time for test preparation, planning, and organisation as the sequential-development software projects, and neither the twelve agile principles or the agile manifesto explicitly states how software testing should be conducted.

1.1 Motivation

Popular testing activities within agile methodologies such as test-driven development, test automation, and continuous integration have been identified and proven to be effective. Technological aspects of software testing, such as new testing tools, techniques, and frameworks, have also been presented in recent years. However, through my searches in the academic literature , I found that there has been little research conducted on the qualities of the software testers in recent years. The ISTQB syllabus - Foundations of Software Testing - mentions only briefly the psychology of testing [8]. As such, there is a need for research on the human aspects of testers working in an agile environment. This thesis therefore investigates the human factors and their combinations that are seen as relevant by the practitioners for the software testing role.

1.2 Research Area and Question

This thesis is a concatenation of systematic literature reviews, an ample qualitative study, comprising of a series of interviews, and a case study, as well as the formation of a grounded theory on the human aspects of the software testing role. The purpose is to understand the desirable human factors and skills for the software testing role, as well as the way they shape each other, as viewed by both software testers and other roles within the development teams. As such, those we interviewed were both testers as well as non-tester personnel, in order to create the full picture of the software tester's role. In addition to the interviews, I have also conducted a case study by observing a project involving a cross-functional team

using an agile development methodology. Since the software testing role was the main focus of my research, I focused primarily on the software testers during my time of observation - taking additional notes, such as the individual's behaviour and interactions. The two literature reviews I conducted, which are to be described further in the thesis, confirm that little investigation has been conducted within this topic. The scarcity of existing research led me to construct the following research question:

RQ: Which human factors and skills are perceived as determinant for the agile software testers role, and how can they shape one another?

1.3 Thesis Structure

The rest of the thesis is structured as follows:

Section 2: Background provides contextual information used in framing my research work, in the scope of the thesis. Since the thesis over-arches across two research areas, the background work on *agile software development methodologies* and *popular agile software testing techniques* are presented in this chapter.

Section 3: Software Testing In Agile Development presents the first literature review conducted, to present a summary of the past ten years within software testing in agile development. The outcome of the first literature review is also used to demonstrate the lack of research on the tester-role human factors within agile testing.

Section 4: Human Factors In Software Testing presents the second literature review, where the identified gaps in the previous chapter are tested. This literature review broadens the summary of human-factors academic research, and looks for literature on human factors in software testing, regardless of development methodologies.

Section 5: Research Method outlines the research method of the Thesis, research design, data analysis approach, data collection procedure, different data types, and validity of the results.

Section 6: Research Context presents both the organizational and project context in which I gathered my data from, in order to familiarize the reader with the research settings.

Section 7: Results elicits my findings from data analysis and data collection, and attempts to present an introductory theory grounded from the data.

Section 8: Discussion aims to compare my findings with the relevant, existing research literature and discuss my results and theory in light of those. Implications for both practice and theory will also be presented.

Section 9: Conclusion summarizes and concludes the thesis as a whole, and elicits contributions to existing works, as well as potential future research within this interdisciplinary topic.

2 Background

This section presents relevant software development methodologies used in agile development. Following, the currently popular agile software testing practices are presented. Both sections are aimed to give the reader a frame of understanding for the research-object of the thesis.

2.1 Agile Development Processes

The agile approach is based on the four fundamental values and twelve principles found in the Agile Manifesto. Process models such as Scrum, Kanban, and Scrumban all emphasize these guidelines. Their iterative and incremental methods characterize agile models. An agile project is usually broken into a series of small cycles. During each iteration/cycle, the team aims to deliver a subset of the functionality agreed upon with the customer. Sequential, single-function teams are not viable for such a process. A rethinking of team structure was needed, due to fundamental activities being interleaved rather than separated. Autonomous cross-functional teams became the essence within agile processes [2], as these teams included people who possess primary specialty within one field and secondary specialties within several.

Compared to sequential software development methods, agile process models with their distinct activities offered an increase in both quality and productivity [12, 42]. It is noteworthy to mention that the root of agile began with Extreme Programming (XP), which has contributed a significant amount to the agile community with its development practices. Some of the commonly known techniques used in agile frameworks mentioned above include test-first development, continuous integration, and refactoring. The current section presents the background work on both software development models and agile software testing techniques. As the thesis is mainly focused on agile development, only process models within the category will be elaborated. The aim is to give the reader an insight into the history and current information on the subject.

2.1.1 Scrum

Scrum is a framework based on iterations and delivering product increments. Projects using this framework are broken down into a set of manageable and understandable cycles that both the stakeholders and the development team can relate. Its work-flow is based on time-boxed series called sprints, which usually last around 2 - 4 weeks. Time-boxed signifies that the sprint ends on a specific date with non-extendable deadlines. There are different roles and groups within the agile framework including, among others, cross-functional teams, the backbone of Scrum. These teams operate with high cohesion and possess dif-

ferent functional expertise within in order to deliver a potentially ship-able product each sprint [15]. Knowledge sharing is high between team members: for instance, a UX designer may assist with software testing tasks if needed. Other roles within the team include Scrum Master; these facilitators guide cross-functional teams and the Product Owner (PO) in the right direction and assist them on the values and practices of Scrum [15]. The role PO represents the customer's side. They may introduce the Product Backlog artifact - a list of items or features that the customer(s) wishes to be implemented into the product [15]. In Scrum, there are so-called ceremonies that occur during a sprint. These include Sprint-Planning Meeting, Daily Stand-up, Sprint Review Meeting, and Sprint Retrospective Meeting.

Sprint-Planning Meeting
During the start of a sprint, a sprint-planning meeting is usually held with the PO, the Scrum Master, and the team. The PO and Scrum Master review the items held in the Product Backlog and re-prioritize them if necessary, and they determine the goals for the sprint. The team may select items from the product backlog (usually from a top-down approach) and into the sprint backlog - a list of items and features which the team aims to complete by the end of the sprint.

Daily Stand-up Meeting
During each day of the sprint, a daily stand-up meeting is held. Its duration should not exceed beyond 15 minutes [76]. Within the time-frame, each team member should answer the following questions:

1. What did I do yesterday?
2. What will I do today?
3. Are there any impediments?

Studies show stand-up meetings are commonly being used for reporting what had been done rather than what will be done. According to Stray [75], the time devoted to each of the three questions are as follows: Question 1: 53%, Question 2: 25%, Question 3: 22%. Stray et al. [76] proposes improvements to stand-up meetings by both removing Question 1 as well as making each member present in the meeting to be standing instead of sitting down, in order not to exceed the allocated time frame.

Sprint Review Meeting
During the end of a sprint, a review meeting is to be held between the PO, the team, and the Scrum Master. The purpose of this meeting is *inspect and adapt* the product - meaning learning and evolve based on feedback given. The PO learns what is going on with the product and the team, while the team learns what is going on with the PO and the market. The review meeting usually includes a demo of the product.

Sprint Retrospective Meeting
The sprint retrospective meeting is about *inspect and adapt* of the process. The team learns what is working and what to improve for the next sprint. Retrospective meetings are useful for making visible bottlenecks as well as areas for improvement [3].

2.1.2 Kanban

The Kanban process utilizes many of the concepts from Lean but is focused more on software development. Kanban principles include: Visualize the workflow, limit Work-In-Progress, Measure and manage flow, and improve collaboratively [1]. As such, it is often used in conjunction with a tangible board and pull-based system [51]. Ahmad et al. argue that the fundamental principles of Lean are mainly overlapping with the principles of Kanban. Agreeably, both Kanban and Lean reflects on characteristics such as working towards flow, develop skillful individuals, increase product quality while minimizing waste. In order to visualize the workflow, a tangible board is used. *The Kanban board provides visibility to the software process because it shows the assigned work of each developer, clearly communicates priorities, and highlights bottlenecks.* [1]. Compared to Scrum, Kanban is task-boxed. Meaning it does not emphasize time deadlines as much as number-of-tasks-done deadlines. Kanban focuses on the completion of prioritized items as opposed to Scrum's how-much-can-we-do within the time limit of the sprint.

2.1.3 Scrumban

Scrumban is a methodology based on both Scrum and Kanban. Scrumban uses several Scrum practices, such as Product Backlog and its ceremonies combined with Kanban practices such as visual management and its pull-based system. According to [50], quoted by [79], *"Scrumban is best used for projects with frequent changing user stories or programming errors, as time-limited sprints of the Scrum model are of no appreciable use, but its daily meetings and other practices can be applied, depending on the team and the situation at hand"*.

2.2 Agile Testing Practices

2.2.1 Early Involvement & Task Delegation

A noticeable divergence from sequential development methods entails involving testers right from the beginning of the development cycles. One of the seven testing principles states that quality assurance activities should be started as early as possible in the life-cycle in order to avoid additional cost and time [8]. In agile, early involvement of testers at

the start of the project, or iteration, enables discussions about user stories or the system architecture to which primary testing personnel will acquire a better understanding of what to test. It also allows them to identify different test environments and scenarios early on - thus increasing the overall productivity and test validity. The delegation of testing tasks is also an essential feature of agile. It is expected that developers perform unit-tests on their own. The encouragement of having multiple roles in a team ensures that developers may also do other testing tasks, such as reviewing user stories and creating test conditions. Dedicated testers, therefore, have more time to focus on other techniques such as exploratory testing, usability testing, and improving test coverage with the developers [68]. Knowledge transfer also happens more frequently and naturally amongst developers and testers, as both testing and development happen concurrently during each iteration. While new team members may benefit from the active feedback, testers also ensure that no misunderstandings or confusion occurs during development. The tight coupling between testers and the rest of the team enables fast learning and greater understanding of each other.

2.2.2 Test-Driven Development & Test Automation

Test-Driven Development has its origin from Extreme Programming and interleaves testing and code development. Its basic principle is to create the tests before implementing code, and its purpose involves producing simple, more cohesive, and less coupled code compared to the more traditional ways [43]. Additionally, the popularity of agile methodologies is due to the assumption that quality software may be produced with a limited workforce. This is achievable with activities that enable automate engineering in a software test process - also known as test automation. Regression testing - the art of executing tests to confirm that defects fixed did not introduce new incidents in other areas - are most efficient when automated [29], thus reducing time compared to manual conduction and increases efficiency. As the code is developed in small increments, the need for recurrently running previous tests, to make sure a modification has introduced no defects to the product, is crucial. Automated regression tests, therefore, can drastically reduce the time and cost of the project.

2.2.3 Continuous Integration

The introduction of multi-skilled workers in agile highlighted a possible nuisance, and several team members may concurrently edit the source code. Old Version-Controlled Systems (VCS) defaulted to strict locking - meaning only one person could edit the source file at a time. With the introduction of agile, new VCS system enabled optimistic locking - meaning several people could edit a source file concurrently. While it is safe to assume that developers need to spend some amount of time with merge-conflicts due to the new locking system, continuous integration was seen as an improvement to the old ways. The continuous integration practice entails that developers should commit small, often, and daily [58].

Integration happens frequently, with small-increments rather than the big-bang method of sequential development models [8]. For some VCS tools, if the source code pushed to the VCS fails to compile or successfully integrates, the tool will notify the developer. Fixing a failed integration is the one of the highest priority in agile development [42].

3 Software Testing In Agile Development

In order to investigate the current research within software testing in agile development, I conducted a literature review. To name a few courses I took before starting my work on the thesis: Software Testing, System Development, and Smart processes and Agile Methods in Software Engineering. The curriculum books and the lecture slides served as a guideline for understanding concepts such as software development approach, agile methodologies, and the different aspects of software testing. According to Kitchenham, a systematic literature review is *identifying, evaluating, and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest* [49]. By conducting a literature review, I aimed to gather as much relevant information concerning the topic as possible.

An existing systematic review regarding the topic, conducted by Fortunato et al., was found: *Quality Assurance in Agile Software Development: A Systematic Review* [28]. Their systematic literature review included academic studies published between 2001 and 2015, and their research questions aimed to answer *which practices are used for quality assurance in agile projects?*, and *What are the main challenges and limitations of quality assurance on agile methods?*. I attempted to replicate the results using the same search string and databases, which proved to be inconsistent with the results they published as some of their findings did not appear in mine. In addition, their literature review had no keywords for software testing in their search string. I sent an e-mail to the authors, asking for elaboration, but a reply has yet to be received at time of this writing. My literature review takes inspiration from Fortunato et al. [28], but it is not a continuation or replication from theirs.

I explored several scientific literature databases for potential literature, as presented in Table 1 on the following page. The databases SpringerLink and Wiley Inter-Science Journal were excluded, as Scopus subsume results from both. Table 2 on the next page shows the search string. The string was modified to fit the syntax of the different databases - however, semantics were ensured to be consistent. A worthy mention is that I included the term 'quality assurance' since there are research papers that refer to software testing activities as quality assurance activities, and the two are interchangeable in agile projects to some degree. My view is that 'quality assurance' is a responsibility taken on by the whole team, whereas 'software testing' refers specifically to the tester role and the specific activities. Nevertheless, both concepts have been included in the search string, for a broader catch.
As Kitchenham suggests, selection criteria are *"intended to identify those primary studies that provide direct evidence about the research question"* [49]. They should also be decided before study selection in order to reduce the likelihood of bias. I devised protocols that include both inclusion and exclusion criteria. They are as follows:

Databases	Link
ACM Digital Library	https://dl.acm.org/
IEEE Xplore	https://ieeexplore.ieee.org
Science Direct	https://www.sciencedirect.com/
Scopus	https://www.scopus.com/

Table 1: Research Databases used for Conducting Literature Review

(”agile software development” OR ”agile development” OR agile) AND (”software testing” OR ”quality assurance” OR ”QA” OR ”SQA”)

Table 2: Search String for Software Testing in Agile Development

Inclusion Criteria

- Relevant academic and industry studies
- Relevant conference, journal, and workshop works
- Review articles, research articles, book chapters, and conference papers
- Qualitative / quantitative research studies published between 2009 - 2019

Exclusion Criteria

- Non-English contributions
- Contributions not related to Information Technology
- Encyclopedias, prefaces, book reviews, case reports, correspondences, discussions, interviews, tutorials, editorials, mini reviews, news.

The selection stages consist of four steps, as presented in Table 3 below:

Stage 1	Identify studies matching search string, inclusion criteria, and exclusion criteria. Discard duplicate studies.
Stage 2	Exclude studies with no mention of quality assurance or software testing and agile in either title, keywords, or abstract.
Stage 3	Review abstract, discarding irrelevant studies for research topic. Conform - and discard - secondary studies with current literature review. Review of introduction, methodology, results, and discussion.
Stage 4	Assess full papers, snowballing references for further study.

Table 3: Selection Stages

Database	Stage 1	Stage 2	Stage 3	Stage 4
ACM	96	7	6	5
IEEE Xplore	262	45	27	21
Science Direct	1344	10	5	5
Scopus	492	29	19	17
Total	**2194 results**	**91 results**	**57 results**	**48 + 2 results**

Table 4: The Results at each Stage of the Selection Process

3.1 Selection Process

I began the review by applying the search string onto the four scientific databases. The application of inclusion criteria, exclusion criteria, and the removal of duplicated studies resulted in a total of 2194 articles during stage one. All articles were skimmed through, excluding the ones that did not mention any form of agile methodology or quality assurance/software testing in the title, abstract, or keywords. The exclusion eliminated over 95% of the initial total - resulting in 91 works. Furthermore, I carefully interpreted the abstract of each article. Studies without their primary focus on software testing were excluded. Studies that did not write about software testing used in conjunction with agile practices or present new agile testing practices were also eliminated. I discarded secondary studies that provided no actual empirical data. However, I used them as a checklist in comparison to my current literature. They also proved useful candidates for snowball sampling. The exclusion eliminated over 30 studies from the selection pool. Finally, each of the 48 papers was carefully assessed. By using snowball sampling, two additional papers were found to be relevant for the review. During the readings of the 50 papers, I noticed a particular pattern of what each article was about. Some entailed evaluating existing techniques within a specific context, while others made comparisons throughout their paper. There were also papers concerning existing activities with innovative mechanisms. I, therefore, began to group research papers which exhibited similar themes. I produced three main categories in which each article was classified and assigned to.

3.2 Three Categories of Agile Software Testing Papers

Figure 1 shows the result of category aggregation. Each research article have been given a classification of either innovative, comparative, or evaluation - each explained respectively in table 5 on page 13.

Six articles were hovering between two categories [21, 17, 63, 56, 6, 34]. These were carefully read and placed into one specific category to the best of my ability following the articles' main focus points. The result of the aggregation shows that most of the papers are an evaluation of existing software testing techniques, strategies for choosing appropriate

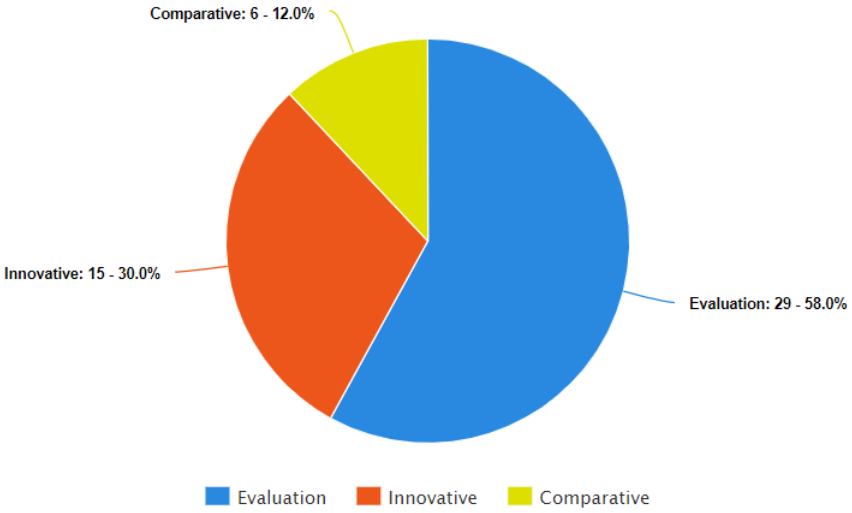

Figure 1: Results Grouped into Patterned Categories

methods or attributes to improve overall quality in conjunction with agile development. Yet, there is no scarcity of papers researching innovative techniques or tools to be used in agile testing, as one-third of the 50 results fall into that category.

The categorized papers were furthermore studied to see the research methodology used. Figure 2 shows that the majority of studies were conducted using a case study. There may be several reasons for why such an empirical method is the favored research methodology. One is due to the industry and research field being tightly integrated in the field of software testing and agile methodology. A case study allows the investigation of a phenomenon within its real-life context, and especially appropriate for theory building and theory testing. Evaluation of testing tools, methods, techniques cannot be realistically measured in terms of usage in an experiment. Seeing how experiments are the only research strategy that can prove causal relationships and provide high precision in measuring outcomes and data analysis, it is reasonable to think why experiments are a significant portion of research methodology used in the innovative category

Category	Explanation
Innovative artifacts	Studies modifying existing ceremonies/techniques/processes which directly/indirectly affect software testing activities in agile development are mentioned here. In addition, papers presenting new ceremonies/techniques, etc. also fall under this category
Comparative studies	Papers falling under this category are comparative studies. For example, the difference in software testing techniques between plan-driven processes and agile processes
Evaluation papers	Papers subsumed under this category include the evaluation of existing known methods/technologies used. Skills required to perform specific tasks related to software testing. Evaluation of testing usage from foreign companies. Challenges regarding utilizing a specific artifact in agile development.

Table 5: Three Categories Generated from Findings

3.2.1 Innovative Artifacts

Innovative artifacts can be divided into Models, Practices, Tools, Processes, and Others, seen in Figure 3. There were two models created in order to assist in agile software testing. A proposed reference model to measure maturity level similar to that of CMMI [72], and a proposed process model focusing on software quality assurance called Agile Quality Assurance Model (AQAM) [39]. There were four practices identified as been developed / modified, these concern new techniques for continuous integration [42], new agile ceremony with purpose of finding and reporting defects [78], automated regression testing [44], and new techniques for creating test cases [77]. In addition, tools for UX-testing have been developed, and specific testing tools were creating for context-specific applications [35]. There have also been developed tools aimed at testing cloud applications [56], as well as creating an E-learning system for agile software development [7]. The two processes entail modification/overhaul of agile process models, one being integrating agile testing with the V-model [4], and the other modifying Scrum to incorporate new testing tasks when developing safety-critical systems [34]. The category 'Others' include different subjects that had no aggregation of more than 1, such as evaluation of new tester role [53], new testing metrics [47], approach for simulation testing [70], or framework for model-based testing [54].

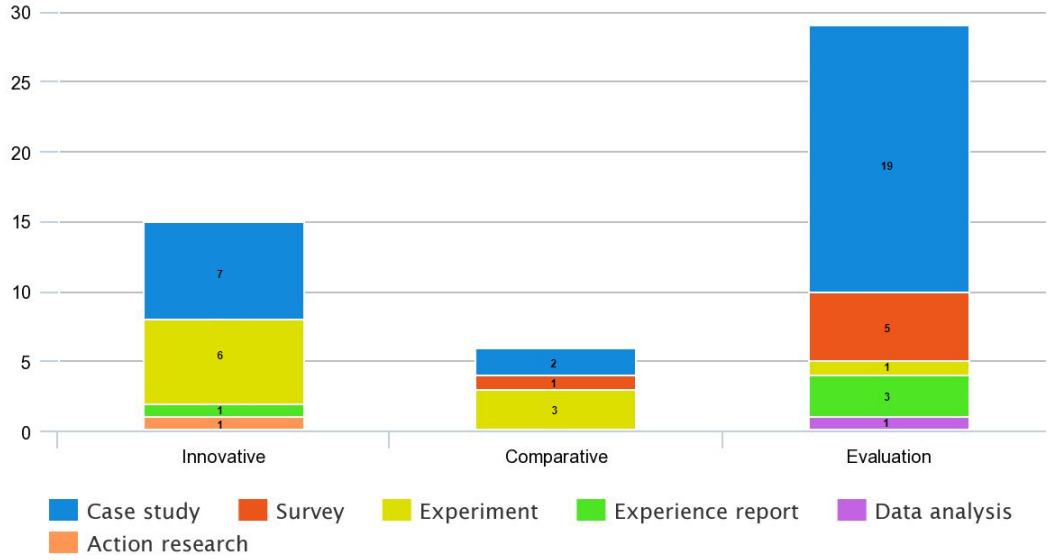

Figure 2: Research methodology Utilized Sorted by Category

3.2.2 Comparative Studies

Two studies were focusing on Test-Driven Development vs. Test Last [40, 60]. In essence, both findings show that Test First-programmers spent a higher percentage of time on testing, a lower percentage of time coding. However, the benefits of test-driven development are small compared to test-last as products delivered were neither higher quality nor were more productive. Nevertheless, using TDD endorsed better branch coverage. Two studies were focusing on the difference of test activities conducted in an agile project compared to plan-driven projects [48, 24]. Kettunen et al. initiated a study criticizing the absence of guidance in agile methods on how testing should be arranged in parallel with development [48]. The study thus raises the question of whether testers are needed in agile development at all as testing is emphasized on developers . The study conducted by Dhir and Kumar [24] compared the process of conducting testing on a web application in a sequential environment and agile environment. Results showed that there was an improvement in the agile model compared to the sequential model. The agile model had increased test coverage, but also reduces cost and improved productivity of the team. Similarly, the study investigating whether dedicated testers are needed or not in an agile context reported that it entirely depends on the team [62]. In order for the team to be efficient without testers, they need to feel fully responsible for the software, have sufficient freedom to decide and control all relevant aspects of their work, and have common ground with users of the software in order to properly interpret the feedback. Disadvantages include integration testing beyond the team level becomes harder to conduct. The last study compared motivation factors for testers [21]. The results are that sequential testers have a higher degree of stress but

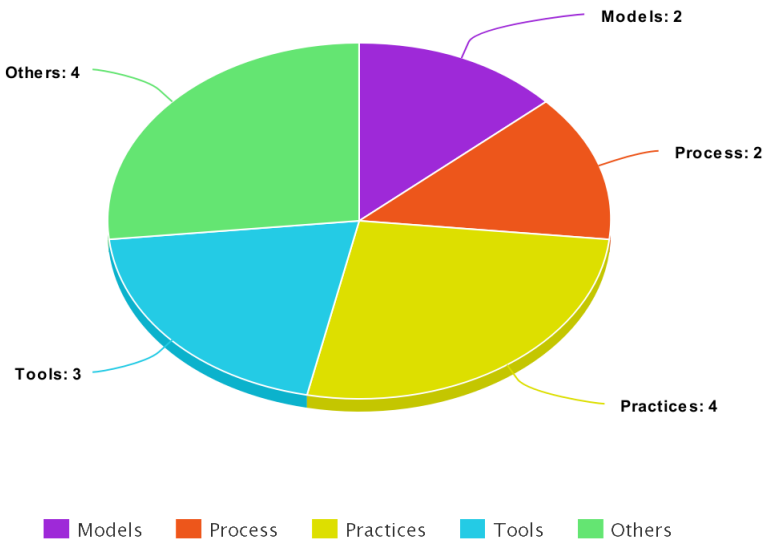

Figure 3: Categories Within Innovative Artifacts

a more positive attitude for tacking challenges. In contrast, agile testers are better integrated into their teams but expressed unhappiness about their relationship with developers as they found it difficult in communication with them.

3.2.3 Evaluation Papers

The 28 studies in the 'Evaluation' category had an immense amount of subjects, Figure 6 shows a total of 13 categories. The summary of each article within this category has refrained from being written. Rather, an explanation of each article can be found in the literature review excel sheet - which I have uploaded on figshare. There are some subjects requiring explanation. 'Perception' indicates that a study focuses entirely on the subjective opinions of agile testing within a company. 'Foreign Usage' evaluates the different agile testing techniques in non-western countries. Although teamwork focuses on the interaction between team members, it can be an underclass to Human Factors which focuses on the soft skills required of a particular role. The reasoning behind splitting up the two is due to scope and focus. Studies placed in teamwork do not have personal skills as their primary focus, whereas studies placed in human factors, does. Attributes for success indicate what factors are essential to consider in order to have the highest possibility of a successful project / deliver a product of the highest quality possible. Assessment deals with the adoption of existing techniques and measuring their effectiveness.

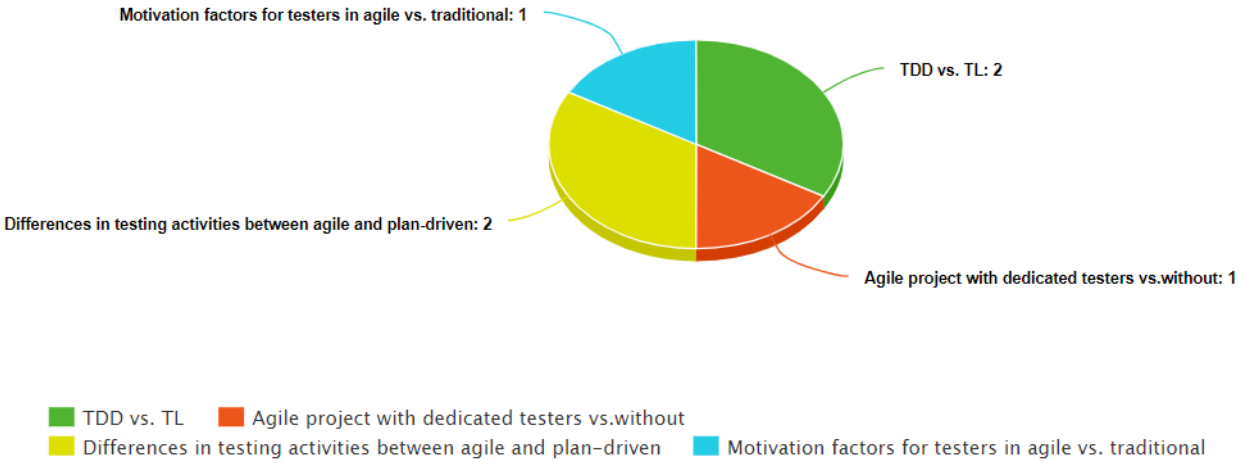

Figure 4: Categories Within Comparative Studies

Category	**Amount**	**Examples**
Quality attributes	7	Variables to improve quality and productivity
Test automation	4	Test automation strategies / challenges in Scrum
Process model transition	3	Transition from plan-driven to agile process model
Assessment	3	Assessing testing methods in an agile context
Foreign usage	2	Testing techniques used in Korea and Pakistan
Teamwork	2	Collaboration among team members to foster testing
Human factors	1	Soft skills wanted by employers, for testers
Exploratory testing	1	Enacting exploratory testing in an agile project
Perception	1	Agile testers' opinion on agile
Security testing	1	Synchronization between agile team and security team
Test-driven development	1	Automated acceptance test-driven development
Usability testing	1	Integration of Scrum med usability testing

Table 6: Evaluation Papers

3.3 Lack of Human Factors Research in the Testing field

The literature review resulted in 50 studies that focused on research related to software testing within an agile environment. Over half of these focused on evaluating existing known methods or technologies used within their respective context. About one-third of the studies focused on researching new constructs in order to assist and improve the software testing field, and 10% of the studies focused on comparing known techniques such as Test-Driven Development with test-last development - the categories from all these studies varied considerably. The majority of the research relates to covering the performance of applications of some testing techniques, automation benefits, or differences of practices applied in project contexts that follow sequential or agile development methods. Studies dealing with human behavior within the field of software testing are scarce, even with the introduction and focus on agile methodology highlighting the importance of people, teamwork, and communication.

The lack of such research in human factors within the testing field enhanced my speculation that the topic at hand is under-researched and fueled my motivation to focus my thesis on human factors - for several reasons. First, the literature review shows that little research has been conducted within this area. The Master's Thesis would, therefore, benefit the research field by building a foundation for future work regarding human factors in software testing. Secondly, human factors should be considered crucial in order to build useful theories and results in other topics. They should be considered, as there may be risks of producing results that do not uncover the key factors when determining the success or failure of a research project. For example, in an agile software project where test automation improvement is needed, reluctance to change might be more important to consider than which automation strategy to apply. Thirdly, supposing that research about human factors within software testing has been conducted, this literature review shows that they have not been researched in conjunction with agile software development. It would, therefore, still be of relevance to research the human factors required/desirable for testers within agile software development.

4 Human Factors In Software Testing

I decided to conduct another literature review regarding human factors and software testing outside of agile development. It is to ensure that the research gap is present and that human factors in software testers have generally not been researched to an extent. Therefore, the main goal of the second literature review is to find research related to human factors within software testing and confirm that scarce research has been conducted within my chosen topic. The secondary goal of conducting another SLR is to highlight current literature within the topic and investigate research gaps for further study. The importance of highlighting human factors applies to both researchers and practitioners. For practicing software testers - if human factors are the determining factor for a successful project - then identifying, motivating, and rewarding testers possessing such factors are vital. For researchers, if human factors prove to be more important than techniques and tools - a new research topic arises. Identifying the most critical factors, and develop ways to identify individuals who possess these factors, as well as determining if these factors are teachable.

The second literature review is consistent with the first one. Scientific databases, inclusion, and exclusion criteria were ensured to be as close in resemblance as possible with regards to the first literature review. Table 7 shows the search string to be used on the four databases presented in table 1 on page 10. Figures regarding protocols and selection stages were refrained from creation, as they are nearly identical as table 2 on page 10 and 3 on page 10.

(”human factor” OR ”human factors” OR ”soft skills”) AND (”software testing” OR ”quality assurance” OR ”QA” OR ”software quality assurance” OR ”SQA”)

Table 7: Search String for Human Factors

4.1 Selection Process

Database	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**
ACM	35	1	1	1
IEEE Xplore	60	15	9	9
Science Direct	956	3	2	2
Scopus	109	3	3	3
Total:	**1160**	**22**	**15**	**15**

Table 8: The Results at each Stage of the Selection Process on Human Factors

By doing the same protocol as before, I began the review by applying the search string onto the four databases. I applied the same inclusion and exclusion criteria and the removal of duplicated studies. The search on the four databases yielded over 1000 results, most of them stemming from the Scopus database. I quickly skimmed through the articles by title, abstract, and keywords, removing the ones who did not mention human factors and software testing. The majority of the 1000 results found were irrelevant, as most of them did not focus on human factors. The ones that did, focused not on software testing, and belonged primarily in the medical and psychology field. A total of 22 studies were selected for stage 2, and was carefully read. The research papers which were not complete or did not focus on human factors within the field of software testing were further excluded from the selection. This literature review identified 15 studies to be of relevance. In addition to the two research articles focusing on human factors from the first literature review, there are a total of 17 relevant studies from the period 2009 - 2019 that focuses on human factors in software testing.

The 17 research papers were carefully read, taken notes of, and checked references to see if any additional papers complied with the inclusion criteria, and which were not found by the search string. There was no additional literature found using this technique. However, a notable trend was that research papers that were recently published cited the same existing research papers within the pool. The research papers that got cited a lot conducted their research based on literature regarding human factors within software engineering in general, and had no specific focus on software testing. Based on this finding, I felt that the literature within the topic of human factors in software testing during this period have mostly been caught.

4.2 Four Categories of Human Factors Papers

Following the literature review in the previous section, I grouped the research papers that exhibited similarities. I aggregated a total of four categories, these include: (1) software testing as a profession, (2) motivational factors for software testers, (3) personal characteristics of software testers, and (4) a miscellaneous category. The lastly mentioned category includes research papers that comprise of different subjects and had no aggregation of more than one. Figure 5 on the following page shows the assigned categories. Note that there are two papers belonging to two categories. Unlike the previous literature review, the contents from the papers were tangible, which proved difficult to assign them into one distinctive category. Each of these categories, as well as the research papers belonging to that category, are elaborated in the next section.

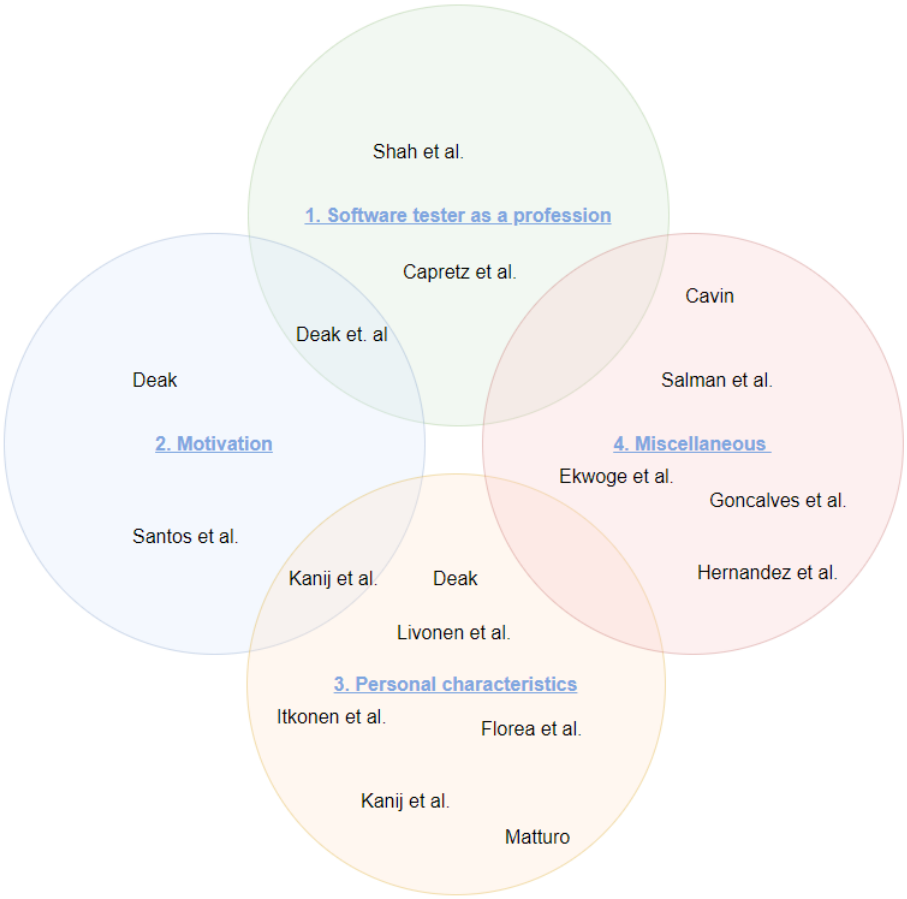

Figure 5: Four Categories of Investigated Findings

4.2.1 Software Testing as a Profession

Studies have been conducted on investigating software testing as a profession. Capretz et al. [13] conducted a quantitative survey amongst professionals in four geographic regions, to find out the degree of attraction of the profession. Results indicate that testing was not a popular career option among software professionals. De-motivators include the treatment of ”second-class citizen” and complexities resulting in stressful and frustrating situations. Shah et al. mentioned similar opinions voiced by senior testers in a service-based software company located in India [71]. The majority of the seniors had a negative attitude towards testing and considered it to be necessary. They voiced the profession as secondary tasks or a stepping stone for a developer career. Being involved in testing allows individuals to gain a more in-depth understanding of the product, and thus make a more substantial contribution to the development of the product. This opinion was also reflected in the junior testers, but with a positive attitude, testing helped them learn the system better so

they could become skilled programmers in the future. The study found that all except one participant did not want to work permanently as a tester. In addition, Deak et al. found that agile testers were more unhappy about their relationship with developers than testers working in a sequential model, which was interesting as agile encouraged more teamwork and blurred line of profession [21].

4.2.2 Testers' Motivation

Studies have also been conducted regarding the motivation of software testers. Mainly, research has proposed a set of motivational factors (i.e., what keeps a tester motivated in order to do his/her job) and a set of de-motivational factors (i.e., factors that could negatively affect the tester in terms of efficiency, creativity etc.). Santos et al. point out the importance of highlighting the testing activities as "a set of human dependent tasks," therefore emphasizing the need for research within motivation [69]. They argue that five factors influences the motivation of software testers: acquisition of useful knowledge during work, work variety, creativity in solving tasks, well-defined work with the precise sequence of steps, and recognition of work. The authors highlight the last factor, which has a lasting impact on the testers' motivation as well as an increase in individual productivity and teamwork enhancement.

Deak et al. also report similar motivational factors [23], such as enjoying challenges, variety of work, and recognition such as positive feedback received from both management and developers. Deak et al. also identified de-motivational factors, exemplified by time pressure, delaying/sacrificing testing activities until the end of projects, lack of resources/planning resulting in inefficiency for testers, or lack of influence and recognition amongst the project. Moreover, some of the participants mentioned the tedious routine of some testing activities and the "feeling of boredom" [23], which would furthermore increase the assumption that the profession is unattractive. Kanji et al. also identified motivation as one of the factors crucial in being an effective software tester [45]. However, their research focused more on personality traits rather than motivations alone.

4.2.3 Personal Characteristics

While designers and programmers are constructive in the sense that they design and build something, a tester's job is destructive; they attempt to break the software constructed. Kanji et al. argue that the effectiveness of a tester role is somehow related to their personality. The authors utilize theory from psychology in order to illustrate the personality traits of software testers [46], which includes five factors: extraversion, agreeableness, conscientiousness, neuroticism, and openness to experience. Results from findings indicate that testers have a significantly higher level in conscientiousness compared to other soft-

ware engineering roles. Conscientiousness is closely related to discipline, hard-working, and dedication. In a similar article, Kanji et al. [45] investigates factors affecting software testers for practical testing. Most of the respondents answering the survey agreed that dedication, thoroughness, interpersonal skill, and punctuality are essential qualities. In addition, the researchers argue that dedication - which is closely related to traits like hard work and responsibility - is partly innate but particularly important for software testers. Furthermore, more than 90% of the respondents believed that good domain knowledge is a must-have for an effective software tester - which is also supported by other researchers [41, 55].

Itkonen et al. investigated what knowledge types testers utilized during exploratory software testing, and came up with three knowledge types. Besides domain knowledge, their analysis revealed system knowledge (the act of knowing the system's mechanisms, logic, interactions) and generic software engineering knowledge (knowledge of usability of the system and the ability to interpret error messages) as the three main knowledge types used. Livonen et al. investigated the characteristics of *high performing* testers - characterized by high defect detection rate, in addition to seen as important by managers and testers alike [55]. They found four themes that are important: experience, ability to reflect, motivation, and personality. Experience refers to implementation and domain knowledge, as well as skill. Reflection entails the ability to maintain the big picture and allowing the tester to prioritize important parts of the software. Motivation was seen as necessary as interviewees mentioned the efficiency of testing correlated with how motivated the testers were. The most personal characteristics of high performing testers were thoroughness, carefulness, patience, and conscientiousness - which supports previously mentioned research.

Two studies focused purely on what soft skills are required for software testers [57, 26]. Matturro investigated what software companies in Uruguay required soft skills [57], 43 advertisements were analyzed and frequency distributed for soft skills within software testing. The results were that the majority of the ads asked for skills like communication skills, teamwork, initiative, and eagerness to learn. Similarly, Florea et al. [26] analyzed 400 advertisements specifically for software testers across 33 countries. The most popular traits asked were communication skills, analytical problem-solving skills, team-play, and independent-working skills. Both articles suggest that there is a more definite need for teamwork and communication skills, analytical solving skills, and pro-activeness. These traits have also to some degree been confirmed by other researchers - Deak et al. [22] and Livonen et al. [55],

4.2.4 Miscellaneous

Other articles that did not fit any of the categories above are mentioned here. Salman et al. conducted a controlled experiment on graduate students, intending to find out if testers exhibit confirmatory behavior - also known as positive testing - when designing functional test cases, and whether such behavior increased under time pressure. The findings resulted in the conclusion that confirmatory test cases were present regardless of time pressure; it is therefore necessary to let testers develop self-awareness and increase their dis-confirmatory attitude [67]. However, the graduates were merely limited to designing the test cases and not executing them. It is possible that by executing the test cases themselves, the testers would gain a sense of accomplishment - thereby further motivated them to design dis-confirmatory test cases - which may alter the results of the experiment.

Cavin assessed the viability of military veterans in becoming software testers by initiating a coursework program [14]. Findings show that most veterans possess human factors that were aligned with the characteristics of a tester - such as communication, adaptability, collaborative, persistent, and work ethic. Hernandez et al. investigated experience and the challenges software testers faced in the automotive industry [36]. Motivation was a critical mention, in addition to openness, friendly attitude, and communication skill - since collaboration with other (non-technical) department was needed. The most recurrent problems found were: unrealistic project planning and estimation, reduced time to perform testing, and late inputs in terms of software, hardware, or requirements. These results are on par with findings from Deak et al. [23].

Going back to psychology, Goncalves identified three main human factors for software testers. Cognitive, operational, and organizational [32]. Cognitive aspects include stress, psychological pressure, retention of the information under mental workload subjection. Operational aspects include conflict, receptiveness, and monotony, whereas organizational aspects entail a lack of training, participation, and division of activities. In addition, the study shows that professional testers faced many factors that are discouraging. Such as outdated testing environment, demobilization, and devaluation of testing career, and often seen as a mere extension of development. Likewise, Ekwoge et al. also identified three main categories affecting testers: cognition, conation, and affection [25]. Cognition refers to memory, problem-solving, decision making. Conation includes impulse, desire, striving, which affects testers' goals, motivation, and commitment. Affection elicits feelings and emotion, influencing respect, team belonging, and social factors of a tester. Finally, Ekwoge et al. also mentions that a tester's job is often destructive but did not mention that the tester role is related to individuals' personality [46].

5 Research Method

This section presents the type of research method I have chosen and the reasoning behind it. I elaborate on the research design and plan for the thesis, and also present the process of data collection and how data analysis was performed.

5.1 Qualitative Research

When conducting a study, it is important to consider which research method to use. There are two main approaches to research, qualitative and quantitative. Qualitative research methods focus on discovering and understanding things in their natural settings, as well as the experiences and perspectives of individuals within that context. In anthropology and sociology, it is commonly known as the *Verstehen approach* [59]. Robsen et al. [65] states that research data gathered within this research spectrum are typically non-numerical, often in the form of words. The qualitative research process includes creation questions and procedures, collecting data from a context-specific environment, data analysis, and the researchers' interpretation of the data. The method of choice within this research category includes observation, interviews, action research, and case studies. The second research type is known as quantitative research, which attempts to maximize the generalizability and replicability of findings [38] - often using methods such as surveys for the former and experiments for the latter. Quantitative research methods may analyze causal relationships between variables, and data gathered within this research type are often numerical and analyzable using statistical procedures in some sense [65].

The qualitative research approach seems to be the appropriate fit for this thesis. Robson and McCartan divides research design into two types, *fixed* and *flexible* design [65]. A fixed design requires attentive planning of the design, which may consume considerable time before data is collected, while a flexible design does not require fixed order of conduct in advance and is liable to change as the study proceeds. As this study may have uncertainties tied to it, the flexible research design was therefore suitable. Flexible research design is commonly used for qualitative approaches, such as case studies, ethnographic studies, and grounded theory studies [65]. Additionally, Ritchie and Lewis state that the use of a qualitative research approach aimed to understand human behaviors in their social and material context [64], which further validates my use of this approach.

5.2 Grounded Theory

The term 'Grounded Theory' first emerged in 1967 by Barney Glaser and Anselm Strauss [31]. It was developed in the context of the social sciences field and aimed to seek to create a causal link from research to theories. Charmaz states that grounded theory consists of

"systematic, yet flexible guidelines for collecting and analyzing qualitative data to construct theories 'grounded' in the data themselves" [16] - hence the name grounded theory. Charmaz also explains grounded theory's nature to go iteratively back and forth between data gathering and data analysis in order to emerge theories from data.

Since its introduction in 1967, grounded theory has gained popularity in research domains such as psychology and anthropology [19]. However, in recent times grounded theory methodology has gained popularity within the field of informatics [37]. Hoda et al. suggest that grounded theory is an appropriate research method in order to *"explore the human and social aspects of Software Engineering"* [37]. This is due to several reasons. Firstly, grounded theory allows for study of social interactions and investigating individuals' behavior - how they interact with the environment around them. This is also agreeable by Badreddin, stating that grounded theory is renowned for its suitability for analyzing human behavior [5]. Since my thesis strongly related to this, grounded theory seems to be the preferable choice. Secondly, grounded theory is useful when studying a relatively little explored area [37]. Both literature reviews in previous sections show that there has been limited research conducted concerning human factors in the field of software testing. Thirdly, grounded theory allows for the generation of concepts and categories which would make practitioners attentive [37] of the content.

5.2.1 Grounded Theory Procedures

There are main procedures for discovering, verifying, and formulating a grounded theory. Note that these steps are not sequentially set in stone during the evolving phase of this study, a significant difference between this type of analysis and other qualitative analysis modes lies in the number of combinations between these procedures in order to develop a sound theory [74]. According to Strauss, there are a total of 10 operations needed for a grounded theory study. A brief explanation is given to each of them.

Grounded theory generates conceptual coding from a set of empirical indicators. These indicators emerge in forms of interview transcriptions and documents regarding some actions and events described by the participants [74]. The researcher uses these indicators in order to derive concepts from them. Strauss calls this *concept-indicator model*, and states that grounded theory is based on such a model. Furthermore, grounded theory's form of *data collection* may be built on a diverse range of data types. According to Charmaz, these include field notes, interviews, and information in records and reports [16]. Data collection never entirely ceases because *"coding and memoing continue to raise fresh questions that can only be addressed by the gathering of new data or the examining of previous data"* [74]. *Coding* is an essential procedure in grounded theory. It denotes the act of naming segments of data - usually interview transcripts - with a small label that summarizes and accounts

for each piece of data [16]. This process is the first step in transforming statements in data into analyzable interpretations [74].

Generation of theories revolves around a *core category*. A core category should amount for *"most of the variation in a pattern of behavior"* [74]. Its prime function is to integrate theory and to render it dense and saturated as relationships are discovered. The iterative nature of data analysis and data collection means that codes are categories are constantly *compared* against each other, and emerging concepts from the study process. This constant comparison ensures that codes can be refined in order to achieve the best fit for data [74]. Grounded theory also utilizes *memos*, a researcher's informal reflection notes intending to capture one's own thoughts [31]. According to Charmaz, memos are useful for constructing analytic notes in addition to *"give you space and place for making comparisons between data and data, data and codes, codes of data and other codes, codes and category, and category and concept and articulating conjectures about these comparisons"*. The motion of *theoretical sampling* elicits concurrent decisions made by the researcher about data sampling and data analysis in order to develop the emerging theory for the next iteration [74]. Its purpose is to *saturate* - elaborate and refine existing categories until *"no new properties emerge"*. Thus, one is able to, *integrate*, and *sort* the data accordingly in order to develop theories.

5.2.2 Reflexive Thematic Analysis

The production of a 'full' Grounded Theory is often demanding and requires larger research projects that are not constrained by time and resources [9]. For the actual data analysis phase, I will, therefore, use reflexive thematic analysis - a somewhat lite version of Grounded Theory with its feature of bottom-up and achievable saturation determined by sample size [10, 11]. Thematic Analysis is a flexible qualitative data analysis approach aiming at identifying, analyzing, and reporting patterns (or themes) within qualitative datasets [11]. It is an umbrella term in the sense that there are different approaches to conduct the data analytic process and not limited to one qualitative approach. I have chosen this method of data analysis due to its strengths; its flexibility means there is no specific research design associated with Thematic Analysis. It is accessible to researchers with little or no qualitative research experience, often more comfortable to learn and use compared to other qualitative analytic methods [9]. The method is also very adaptable in terms of the research question, methods of data collection, and sample size [10, 9], and just like Grounded Theory - Thematic Analysis' form of data collection may be built upon a diverse range of data types, such as field notes, interviews, and information in records and reports [16].

Braun and Clarke divides Thematic Analysis into three approaches, *Coding reliability*

approaches, *Code book approaches*, and *Reflexive approaches* [11]. In coding reliability and codebook approaches, data analysis is typically done through coding by multiple researchers with pre-defined codebook and reaching a consensus level of agreements. Thus, these approaches indicate that data are allocated to pre-identified themes. Coding reliability and codebook approaches follow a scientific method, where researchers move from theory to hypothesis, to evidence gathering [11]. Reflexive Thematic Analysis is commonly inductive as the researcher's subjectivity is often portrayed as a resource rather than an impediment [11, 81]. Themes in reflexive approaches are therefore not pre-defined but built on top of the codes generated from the researcher's interpretation of the data.

Similar to Grounded theory, Reflexive Thematic Analysis provides an iterative way for coding and offers flexibility in the sense that codes may evolve during the process [11]. As an inductive approach is more common in Reflexive TA, it allows the coding to become the building blocks for the themes/theory [9], which matches this thesis' exploratory study. Braun and Clarke defines six phases to conduct a reflexive thematic analysis [9]. I aim to follow the framework as closely as possible.

Familiarization is the first phase occurring after an iteration of completed data collection. The researcher becomes 'immersed' in the data through reading and re-reading the textual data, he/she may make casual notes about potential possibilities and connections between the data gathered, the participant, and existing literature [9]. By 'knowing' the data and noticing things of interest, this phase can significantly help during the next stage.

The second stage is to attach clear labels to chunks of data in order to group them into patterns and which exhibits similar phenomenons - coding. There are two broad orientations to coding [11, 81]. *Inductive* approach denotes working 'bottom-up,' using the data as a starting point to generate themes. This approach does not assume that the researcher has no prior knowledge of the subject - rather, it signals a data-led analysis over one from existing concepts or theories. Another approach is for researchers to analyze the data with existing concepts, theories, and literature as foundations - a *deductive* approach. This approach is commonly used in coding reliability and codebook approaches, hence the usage of a pre-defined codebook. Additionally, the researcher also needs to decide between coding types - *semantically* or *latently* [11, 81]. *Semantic* (or descriptive) codes capture the explicit meaning of the data. In contrast, *latent* (or interpretive) codes capture the implicit meaning - ideas, concepts, meanings - which reside beneath the surface of the data and require the researcher's interpretive lens [9].

Through familiarization and coding, the researcher may begin to form and identify patterns. These patterns can then be used to *generate themes*. A theme consists of a set of

data that exhibits similarities or overlaps each other, providing a patterned meaning of the data [11, 81]. Developing themes is an active, iterative process, meaning themes may be constantly replaced, refined, and rejected as the researcher analyses more data and furthermore, *revises* and *defines them*. Therefore, one should not get too attached to the prototype themes created in the early stages of data analysis [11].

Review phase is a constant, iterative process. It advocates the researcher to ensure that the data within a specific theme 'makes sense,' and whether the themes have distinctive boundaries from each other [11]. According to Willig and Rogers, reviewing makes sure the themes *"work well in relation to the coded data, the data-set, and answering the research question"* [81]. Braun and Clarke introduced thematic map, *"a process of virtually exploring potential themes and sub-themes, and connection between them"* [11] that can be used both when starting to generate themes and for reviewing them at a later date. Thematic maps are useful for checking how themes fit together - whether they work well - and checking that there is no overlap between them [81]. *Defining* themes helps with clarity and cohesion between the themes by renaming them so that each theme exhibits a core idea and meaning. Both of these phases *"ensure that themes, and theme names, clearly, comprehensively and concisely capture what is meaningful about the data, related to the research question"* [11].

Finally, *producing the report* lets the researcher rise 'above' the analytic phase, and see the big picture. The researcher connects all of his findings and notes with existing research and literature on the topic in order to answer the research question [81]. This last phase acts as the final test of *"how well the themes work, individually in relation to the data-set, and overall* [11].

5.3 Data Collection

This thesis is based on the data collected through qualitative measures such as observation and semi-structured interviews. Data collection was ensured to comply with Yin's four principles of data collection [82]. Table 9 on the next page shows how I applied these principles during my thesis.

5.3.1 Interviews

Interviews were conducted throughout the data collection period. These interviews provided a deeper insight into the interpersonal characteristics and behaviors of testers, seeing by the testers and other roles. According to Yin, interviews are to be considered *"guided conversations rather than structured queries"* [82]. Table 10 on page 30 shows the three main categories of interview types. For my thesis, I have selected semi-structured interviews with the most appropriate type.

Principle	**My Approach**
Use Multiple Sources of Evidence	Observation, semi-structured interviews, and conversations logs are used as data sources in order to achieve data triangulation and increase the quality of this thesis.
Create a Case Study Database	Data collected are inserted into a qualitative data tool for analysis. Relevant notes were also created, both analog and digitally.
Maintain a Chain of Evidence	Observations, and interviews were noted with the place and time of the conduct. The data collected were aggregated by type and date. Traceability was also kept in mind during the coding of the data. These measures were taken in order to increase construct validity of the information for this thesis.
Exercise Care When Using Data from Electronic Sources	Data collected from electronic sources were scrutinized and kept in their contextual domain. If possible, they were cross-checked using other sources in order to assess their validity.

Table 9: Four Principles of data Collection

Semi-structured interviews offered to cover most of the relevant topics I was after, as well as including certain flexibility such that additional information that may be relevant, could be discussed further. Unstructured and structured interviews were not preferable, as the former emphasized openness and the latter more on a structured form. I wished for a balance between the two. While keeping the discussion open was important, it was also focal I did not lose sight of the objective during the interview. An interview guide was created beforehand and followed as main guidelines for each interview (which can be viewed in Appendix B, C). Additionally, a report form was sent to the Norwegian Centre for Research Data (NSD) asking for permission to conduct interviews - the report form was approved without complications. Moreover, before the interview, the interviewees were given a written consent form with the information of the interview, the usage of the data, and their commitment during the interview (see Appendix A). Interviewees were notified both orally and in written form that they may withdraw from the interview at any time.

The interviews were allocated a time-slot of 45 minutes. However, the actual length of each interview varied from participant to participant. The interviewees' names were obfuscated in order to ensure full anonymity. A tape recorder was utilized in order to record the interviews. The data were furthermore transcribed for data processing, saved, and linked together with the corresponding participant. Using a tape recorder meant that I could focus

Interview type	**Explanation**	**Reasoning**
Unstructured interviews	Does not contain pre-made questions that are expected to be asked to an interviewee. An unstructured interview resembles a guided conversation between two individuals. The interviewer may probe the topic area and come up with questions on the spur of the moment [20].	This was not suitable for me as accurate, consistent information needed to be extracted within a specified time-frame.
Semi-structured interviews	Adds a certain magnitude of formality in contrast to the above mentioned by introducing predetermined, open-ended questions and predetermined orders in the form of an interview guide.	Semi-structured interviews offers flexibility as the interviewer may ask additional questions emerging from the dialogue [20].
Structured interviews	Known for having little to no wiggle room in terms of open-mindedness [20]. These interviews are strict in terms of questions and guidelines, ensuring that all participants provide the same data given the same predetermined questions [30].	Some degree of openness needed to be incorporated due to the subject of this thesis, I have therefore not chosen to pick this interview type.

Table 10: Interview Types

more on the actual conversation-flow and ask the right follow-up questions instead of frantically taking notes. These were stored at a secure server. The interviews were conducted in Norwegian - as such, so were the transcriptions. Citations taken from the transcriptions were, however, translated into English. They were ensured to convey semantically the same as in Norwegian. Even though transcription was the most difficult part in this process, I gained much insight and reflection on what has been said during the interviews. Existing transcriptions also helped later on as I conducted more interviews, seeing how I could connect the phenomenons from previous transcriptions even before coding.

It is worth noting that interviews were my primary source of data. The sources are members of a software service providing company - which are to be further elaborated in section 6.1 on page 37. These professionals have varying profession and years of experience. I felt that it was important to gather data from non-testers, obtaining their perception of testers in order to understand the complete picture for the study. These interviewees are all currently utilizing agile methodologies in their respective teams, and are listed as follows:

Interviewee software field	**Work experience**	**Interview length**
Software testing	7 months	53 min
Software testing	1 year	1hr 3 min
Software testing	4 years	51 min
Software testing	6 years	1hr 6 min
Software testing	3 years	41 min
Software testing	11 years	30 min
Software development	1 year	23 min
Software development	1 year	24 min
Software development	2.5 years	20 min
Software development	2 years	26 min
Software development	10 years	38 min
Interaction design	1.5 years	37 min
Interaction design	4 years	41 min

Table 11: Overview of the interviews

5.3.2 Observations

I was also fortunate to be allowed to observe the project in which one of the interviewees is working at. Both the organization and the project context are given in section 6.2 on page 37. Yin divides observation into two categories, direct observations and participant-observation [82]. Direct observation entails a passive observer within the environment,

whereas in participant-observation, the observer can interact with the participants. Observations provide valuable data that complement interviews with specific individuals.

My observations were direct, since I tried to minimize my interaction with the team as much as possible in order to minimize the external influence that may impact their everyday work behaviors. However, I was engaged in social discourse with the observed team members, such as making small talks during lunch. Additionally, I became friends with one of the testers and made small-talks during social events. Some information was categorized as useful, which were noted down at a later date and used in order to refine the interview-guide.

I started observing the project from the 1st of November 2019 to January 2020. During the time of observation, these ceremonies were present:

- Observation of meetings
- Non-participatory observation of daily work-flow and team interactions
- Observation in form of casual and formal interactions during work-hour and lunch-hour

Observation of meetings includes daily stand-up, status meetings, and workshop ceremonies. During these meetings, I noted down the participant's identities, their role within the team/company if possible, the duration of the meeting, and the topics being discussed. Additionally, I noted down who was acting as the facilitator, the behavior of the tester during these meetings if present, and personal thoughts that arose both during and post-meetings. I categorized the meetings into different types and noted down the amount of each.
During the work-day, I noted down the timestamp and points of interest should they occur to me. Points of interest include conversations between the tester and another team member, a team member's interaction with a tester, or when a tester asks another tester for help. I also noted down what each participant in the team was doing, with a special focus on the tester of the project. If nothing significant happens, I logged everyone's status every 10 minutes.

When observing casual and formal interactions during the work-day, I followed the same procedure as above - noting down the timestamp and the participants during the interaction. I also documented the topic that was discussed to the best of my abilities. During lunch-hours, I was participating in small-talks, things that strike me as interesting were noted down either in my laptop or in my notebook after the lunch break. All of these procedures stem from the observation protocols, which were constructed under the guidance of my supervisor. The observation protocol can be viewed in Appendix D.

Meeting type	Participants	Length
Daily stand-up	7	8 min
Daily stand-up	7	12 min
Daily stand-up	8	12 min
Daily stand-up	6	8 min
Daily stand-up	8	11 min
Daily stand-up	8	14 min
Daily stand-up	8	15 min
Daily stand-up	8	17 min
Test status meeting	9	27 min
Test status meeting	8	38 min
Domain expert workshop	6	2 hours 40 min

Table 12: List of Meetings Observed

5.3.3 Chat Software

Chat software was also used to extract additional support for the data collected through observations and interviews, such as follow-ups questions or elaborations. The software utilized is called Slack, a chat application aimed for business purposes. Data collected from the logs were data deemed of relevancy and therefore were treated in the same way as data collected through other means. Screenshots of the chat logs were saved and added into the analysis tool NVivo.

5.4 Data Analysis

I attempted to follow the reflexive thematic analysis procedures as closely as I could. After each transcript was done, I saw down and re-read each and one of them, occasionally taking notes on interesting things. I also sat down and wrote notes shortly after each interview, these notes - such as associated ideas, compelling remarks, and interesting quotes - stored valuables that could have been quickly forgotten if not written at once. Furthermore, the interview questions were iteratively refined to reflect on the previous participants' answers. During the initial stage of data collection, I focused mainly on questions that asked participants what he/she thought was desirable for a tester, then gradually used those answers to look for common patterns and continually refined questions in order to probe for more in-depth elaborations.

Before any themes could be established during this study, the raw data had to be processed and analyzed. The data were grouped into different sections; interview transcriptions, observation notes - both passive and meeting-specific - and chat logs. I used NVivo 12 to do most of the coding. NVivo is a Computer Assisted Qualitative Data Analysis Software [66]

(abbreviated CAQDAS). The raw data were uploaded onto NVivo and processed. Braun and Clarke mentioned two main coding approaches, *selective coding* and *complete coding* [9]. With *selective coding*, the researcher only selects the data of a certain phenomenon in which he is interested in analyzing. At the same time, *complete coding* codes all data that could potentially be relevant towards the study, and then becomes selective towards the later analytic process. I opted for a complete coding approach as using the other might make me miss out on 'hidden' phenomenons that have yet to emerge.

Though Clarke and Braun did not explicitly state the optimal coding technique for use in conjunction with Thematic Analysis, I chose to follow Saldana's descriptive coding technique [66]. The technique entails summarizing topics as a single word or a short phrase. The descriptive coding technique is particularly suitable for beginner qualitative researchers, and when the data are embedded in various forms, such as interview transcripts, field notes, and documents. I constantly made folders, and kept my data as organized and systematic as possible. Since the scope of this study is quite large, I had to be wary when naming the codes. However, even when codes were quite varied, they may fall under the same umbrella of a theme. Like Braun and Clarke stated: *"A theme is like a wall or a roof panel of a house, made up of many individual bricks (codes). Good code will capture one idea; a theme has a central organizing concept, but will contain lots of different ideas or aspects related to the central organizing concept"* [9]. Towards the end of coding, I had a total of (around) 120 codes and sub-codes. Some of the codes described a similar phenomenon, but with a small variety. Throughout my data collection and analysis phase, I continuously went through each and one of my codes, grouping and renaming them in different ways whenever possible, and deleting the ones that did not make sense. I ended up with 95 codes and sub-codes in total. Figure 6 shows an example of how the raw data was processed into a code, and then allocated to a theme.

5.5 Validity

In qualitative research, it is imperative to assess the degree of soundness and cogency of a study [82]. In other words, the validity of the research must be explained and examined. There are multiple threats when conducting a qualitative research [82]. Both Yin [82] and Robson [65] identifies four types of validity. *construct validity*, *internal validity*, *external validity*, and *reliability*. Each particular explanation can be found below. Section 8.12 on page 73 explains how I took these validity's into concern, and aims to convince the reader that the trustworthiness of the results are sound, and minimal participant (and researcher) bias have been upheld.

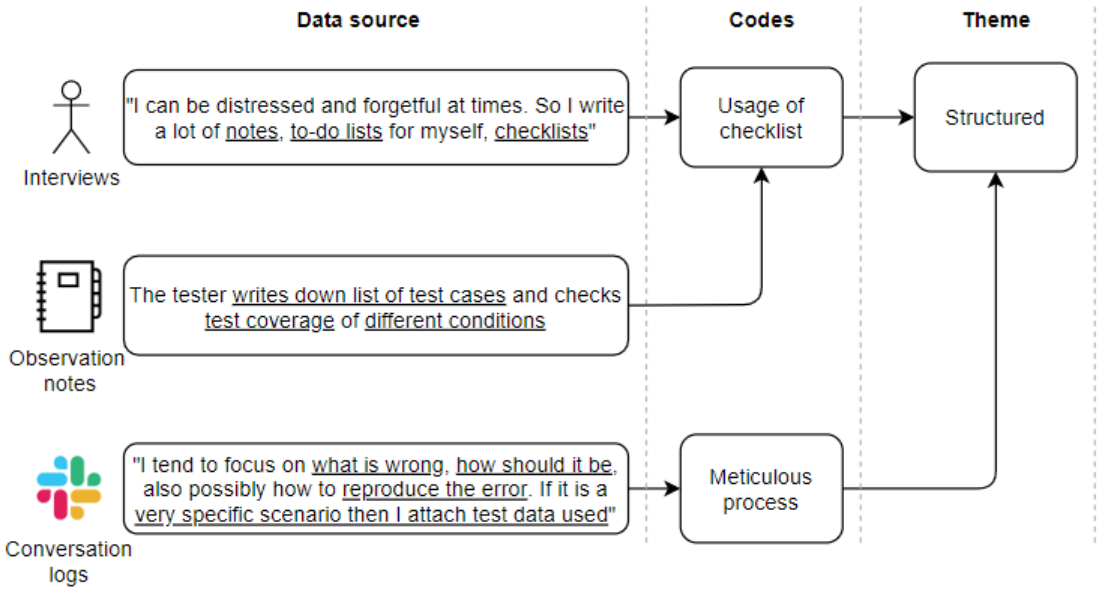

Figure 6: Example of how data was Coded

5.5.1 Construct Validity

Construct validity refers to how well a study measures its claimed construct. It concerns whether the study has been affected by the researcher's subjective judgment [82]. According to Yin, *Multiple sources of evidence*, *letting key informants review draft case study report* are tactics that can be used in order to increase construct validity of a study [82].

5.5.2 Internal Validity

Internal validity aims to evoke the causal relationship between the *treatment* and the *outcome* [65], ensuring that the findings of the research is valid 'internally' to the specific research. According to Robson, common threats to internal validity includes 'history', labeled by Campbell and Stanley [65]. History refers to something that may happen at the same time as the investigation - which may interfere with the interpretation of the causal relationship between the treatment and the outcome [82].

5.5.3 External Validity

External validity, also known as generalizability, refers to whether the study's findings are generalizable beyond the contextual setting of the case study [82]. For case study, special attention is recommended for analytic generalization - that is, striving for generalizable

findings and *"going beyond the setting for the specific case or experiment that had been studied"* [82].

5.5.4 Reliability

Reliability entails that if a researcher at a later date conducts the same study using the same procedure, the individual should arrive at the same findings and conclusions [82]. Yin states that case study follows an analogous logic and that the case study should reflect either a literal replication - where one predicts a similar result, or a theoretical replication - where one predicts conflicting results, but for anticipated reasons [82]. Reliability is, therefore, a concern for the case study due to its complexity for replication.

6 Research Context

In order to get an adequate understanding of the findings that resulted from this thesis, it is principal to be aware of the research context in which the study took place. This chapter presents the contexts, such as a brief description of the organization and the project permitted to observe, as well as the team structure and team members' roles in the project. The elaboration serves to inform the reader well enough to understand the findings presented in section 7 on page 44.

6.1 The Organization

The primary source of my data, the interviews, were gathered from professionals working for the same company, which I refer to as Software Service Providing Company. The Software Service Providing Company is a medium-sized company with over 500 employees in Norway, Denmark, Kyiv, and Bratislava. The company offers expertise within project management, software testing, software development, interaction design, maintenance, security. In Norway, Software Service Providing Company possesses over nine software testers who are spread across different projects.

6.2 The Project

I was also permitted to observe a project in which one of the tester-interviewees is currently employed to. The project takes place in a customer-organization (now referred to as Sierra) prominent within the financial sector. The organization consists of over 1000 employees in Scandinavia and has focused, among other things, on automotive financing, sales financing, and loans. Sierra employs professionals, which assists Sierra in software development, maintenance, and innovation of services. The project consists of one team of professionals, all hired from Software Service Providing Company, working on multiple projects simultaneously.

The conventional single product-development was not at the center of attention within this case. Instead, the group of professionals works as a 'Concurrent Integration Team.' Seeing how the technical architecture of Sierra is modular, the team was therefore working with the integration of different projects with different systems across Sierra. The group had two main projects in focus. Project One entailed information extraction, which is used at a governmental level. Exemplified, if a governmental entity requests information regarding an individual, the - result from the completion of the project - will extract all relevant information about the specific individual. The team was therefore required to integrate several systems in order to achieve the functionality. A simplified illustration of the project can be seen in Figure 7 on the next page, where the team works on the Sierra External Service.

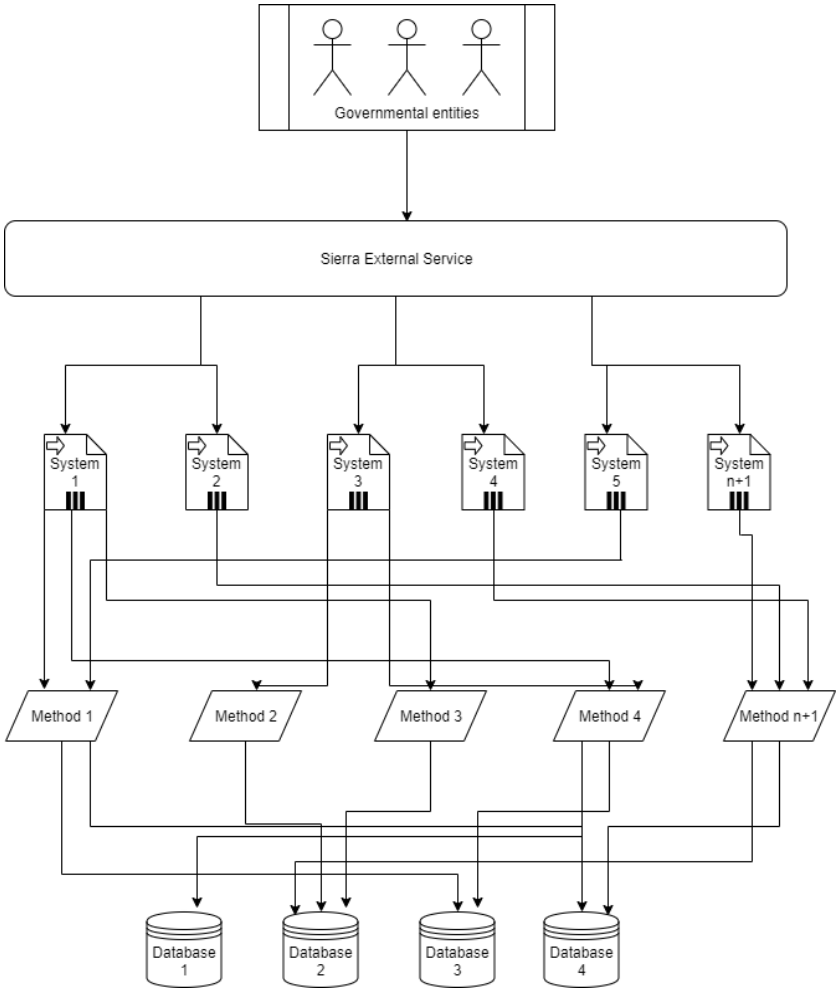

Figure 7: Simplified Technical Structure of the Project

Although Project One was heavily prioritized, the team was also working on Project Two. Project Two was similar in the sense that several systems also need to be integrated, hence the name 'Concurrent Integration Team'. Project Two entails automating the process of information exchange between a consumer loan and a national register. This process is currently done manually and tediously. Project Two, therefore, served to ease the tedious process through automation, saving resources and time.

6.2.1 The Team

The team numbered in a total of eight team members. As the projects are mainly integration and maintenance, there were no typical UX-designers or a dedicated business analyst member within the team. Four members were performing testing activities, and four are

conducting development tasks. It is worth noting that the team members were halfway distributed, since they were divided into two locations. Half the team was located at Sierra's headquarters, while the other half sat together in Kyiv. Note that obfuscated names have been provided to each of the team members.

Name	Role	Location
Amelia	Functional architect / technical test-lead	Sierra HQ
Vanessa	Test-coordinator	Sierra HQ
Eugene	Architect / tech-lead	Sierra HQ
Alice	Test automation engineer	Distributed
Adam	Test automation engineer	Distributed
York	Developer	Distributed
David	Developer	Distributed
Robert	Developer	Distributed

Table 13: Team Members, Their Roles, and Their Placement

The person of interest within this case is Vanessa, since during my time of observation, Amelia acted more as a functional architect rather than a test-lead. However, she also performed testing tasks from time to time. Observing Alice, Adam, York, David, and Robert would prove to be infeasible due to the extensive distance between the team-member and myself. Concerning work experience, both Amelia and Eugene worked on both projects for over two years. In contrast, Vanessa is a graduate that arrived in March, and one of the reason why Vanessa sat next to Amelia. Figure 8 on the following page shows the seating structure of the team during my time of observation. The figure shows an open office area divided into two, one for externally hired professionals and another for Sierra's employees. The team sits together with other professionals from the same Software Service Providing Company, but who are working on different projects. Sierra workers' area consists mainly of technicians, some with excellent technical knowledge with the databases seen in figure 7 on the previous page, which fostered increased communication between Vanessa's team and Sierra workers. Every day around 09:05, Vanessa's team went into the meeting room and, with the help of Microsoft Teams, performed the daily stand-up together with the rest of the team members.

6.2.1.1 Responsibilities

During my observation, I got to know most of the team members' everyday tasks with some exception to the developers. The responsibilities of each team member varied a lot, but were sometimes also overlapping. Even though the team does not have a project leader, Amelia seemed to have assumed the role as everyone respects her decision and values her input. Amelia's work included inspecting the functional architecture, and devise potential areas-of-failure within systems - these are noted down as descriptions for Vanessa. Vanessa's

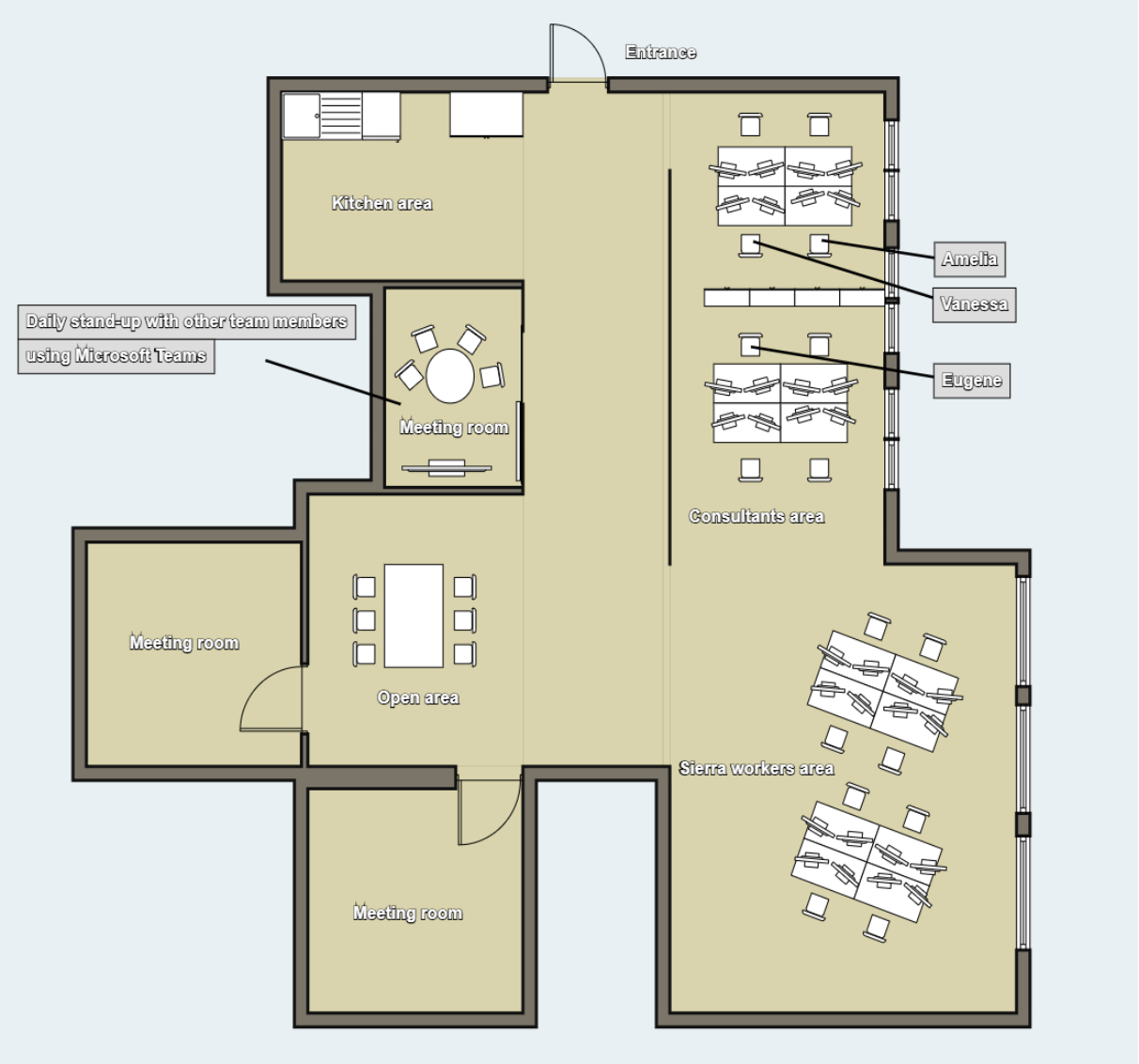

Figure 8: Sierra Seating area

primary everyday task was to create test-cases based on Amelia's descriptions. However, she also needed to check if the descriptions were correct (i.e., validating potential assumptions and verifying potential failures occurring in specific modules). Secondly, since Vanessa is a graduate recently arrived in March, she also had to spend a lot of time learning about the domain, due to her inexperience compared to the rest of her team members, who have worked on the project for over two years.

During my time of observation, Vanessa spent a significant amount of time inspecting the technical aspects and understanding the business domain knowledge. Seeing as how Amelia possesses more experience in both technical and domain knowledge, she acted as a mentor for Vanessa. It was observed that the two communicated a vast amount during work hours. At regular intervals, Vanessa asked Amelia both technical and domain questions. Additionally, when Vanessa found a bug, she forward it to Amelia, who investigated it further before registering it on the board.

Eugene's responsibilities lied in being an architect with the technical overview. He spends lots of time understanding the technical architecture of different systems. Additionally, Eugene acts as a tech-lead for the distributed developers by assisting them, such as performing code reviews and replying to technical questions. Furthermore, he took upon the most challenging and complicated tasks, such as overhauling the data access technology used in one of the projects. Both him and Amelia also spent considerable time in meetings, such as feasibility meetings and weekly meetings with team leaders for the different systems that needed to be integrated. As for the distributed team members, their everyday tasks were less detailed, seeing how I was not able to observe them. However, small talks during lunch with Vanessa and Amelia disclosed some everyday task that Alice and Adam are performing. The test automation engineers' job was to automate all the test-cases created by Vanessa. Therefore, Alice and Adam had more contact with Vanessa than the rest of the team members.

6.2.2 Processes

The team's development process was mostly agile, using Kanban's development methodology combined with some of Scrum's ceremonies. However, according to the team members, there were times when the team was forced to work with a waterfall development process. The development process was, therefore, a combination of agile/waterfall, depending on the state of the projects. During my observation, however, the team was only utilizing agile development methodology. From my observation and discussion with the team members, it appears that the agile approach occurred most of the time when a Product Backlog Item (PBI), usually in the form of a user story, appear on the board, Amelia and Eugene creates development task and a corresponding testing task. As soon as the PBI is complete - mean-

ing both the tasks are marked as developed and confirmed tested - the team releases a new increment. The team operated in the sense that there are no time-boxed sprints but works preferably continuously as backlog items appear. If an update is required for a PBI, the team members will merely add a new task on it, as there is no need to go through a long process of re-evaluation. Also, each working test-case is automated by Alice and Adam, thus enhancing agile development by automation-as-they-go.

6.2.2.1 Daily Stand-up Meetings

The daily stand-up meetings were held every day at 09:00. All of the three questions - as presented in Section 2.1.1 on page 4 - were asked during the meeting. During my observations, it was noted that these meetings' purposes were to update team members on each other's progression and to bring up problems with regards to specific fields. On average, the meetings lasted from five to ten minutes and rarely reached the 15 minutes mark. The meeting was conducted online using Microsoft Teams and video-chat, so everyone was able to get real-time contact with each other.

6.2.2.2 Test-status Meetings

Test-status meetings were held on a bi-weekly basis. Not every team member was included during these meetings, Vanessa and Amelia were the ones participating from the integration team. Testers joined Test-status meetings from different projects. The purposes of these meetings were to establish coordination through communication between projects in Sierra. Each member was able to report on their current test status in their projects, and discuss any issues they are currently facing. Additionally, testers could ask other testers for help. It was observed that during one of the test-status meetings, one tester asked Amelia for assistance with regards to integration testing. Furthermore, these meetings proved to be useful for developing potential countermeasures against common problems. I observed a discussion regarding the common test-environment used by all projects, as the projects are partly dependent on each other, each needs to make sure that data are correct and not manipulated by others during their respective testing.

6.2.2.3 Feasibility Meetings

Since the integration team's main objective is to integrate most of Sierra's systems, they do not have one single product owner and instead, receive feature requests from many different business divisions in Sierra. Feasibility meetings are, therefore, ubiquitous within the team. These meetings - or rather, discussions - often consist of only Amelia and Eugene, since the two have the most experience and insights. The two evaluate feature requests - if it is feasible, Amelia creates a PBI in the backlog with a technical description of possible solutions.

6.2.2.4 Sprint Planning and Retrospective Meetings

The abnormality with this case was the absence of sprint planning and retrospective meetings, the team had neither of those. The lack of sprint planning meetings was due to several reasons. First, the integration team had no product owner. The team, therefore, had a high level of autonomy and was self-governing. Most tasks that appear on the board were tasks that have been approved during the feasibility meeting. Secondly, the team felt like sprint planning meetings were unnecessary due to the effectiveness of daily stand-up meetings - providing a good overview of prioritized assignments. With a specific product owner omitted, the team, therefore, saw no point in having sprint planning meetings. Nevertheless, retrospective meetings have been discarded by the team. One of the team members remarked that the team was not too glad about it, the team is not *"in the mood"* to talk about what it has been doing good, what can be done better, and discussing the current work process. It seemed that the team possessed an 'if there is work, it works' kind of mindset.

6.2.3 Tools

The integration team was using several tools to increase the effectiveness of their work, with the most notable being Microsoft's Azure DevOps. With the team being most proficient in technical aspects, Azure DevOps offered an excellent platform for coordination between team members and across different fields - ensuring consistent overview over backlog items on the visual board between members. Additionally, its integrated version control system and test-plan capabilities allowed developers to become more intertwined with the testers. The team also utilized Microsoft Teams for communication between Sierra HQ's members and the distributed ones. The use of such a tool made it easier to coordinate with the addition of video chats.

7 Results

This section presents the results found during the data collection and analysis stage. Note that a theme could have several sub-themes; as such, a thorough explanation is given to each. It is also noteworthy to mention that some themes are mentioned more often than others from the data, these are not any less essential or insightful since frequency does not determine value [9]. Because interviews, especially semi-structured ones, generate fluid and flexible data, it is thus possible to end up with data variation. Part of the findings presented here are in line with my preliminary results reported in Paruch et al. [61] (Appendix E), while others have emerged from further data analysis.

7.1 Human Factors

This section describes the human factors as perceived by the professionals both working as software testers, and with software testers. Human factors here refers to the 'interpersonal' skills that are applicable to any type of job (e.g. being structured, or creative). Note that it is possible for other skills than the ones described here to appear; the factors presented here are the ones that have emerged upon analyzing my current data-set. These findings are mainly backed up with citations and statements from the software professionals.

7.1.1 Adaptable

Adaptable is the ability for a person to be flexible, respond quickly to change, and adapt to changing work conditions. Three testers mentioned that one has to adapt to oneself following the current situation quickly. One commented: *"Yes, it can sometimes be as early as after I looked over my tasks and feel ready to start, someone would pat me on the shoulder and say that I have to do something else"*, another said: *"We work with prioritization, when an item of high priority is incomplete from the developers' side, I'll start to work on something with medium (priority). However, often the developers finish before I get to complete the testing on that item, so I have to drop it and start testing on the other one (with high priority)"*.

It was further observed that Vanessa from the project often had this issue. She often mentioned that since she was newly qualified, the constant context-switching proved to be quite challenging to keep up with the rest of the team since testers were often the ones that had to deal with both domain-specific tasks and testing the technical aspect of the system. Since she sat together with a more experienced tester, I observed that Vanessa frequently asked questions regarding both knowledge types. Out of all testers interviewed, three commented on the importance of adaptability in correlation with agile software development, stating that this trait was somewhat less of a focus on sequential projects.

According to the majority of professionals, being adaptable was a needed trait due to the constant changes happening in agile environments. During the interview with Vanessa, she said *"I think this is sort of the highlight of someone who is working agile, the fact that one can quickly 'switch' between things and the fact that one has sufficient control so that it does not take long to adapt something different"*. When asked the ones who also have worked sequentially in prior times, the participants preferred the agile way of working. One experienced tester noted that agility improved the team's effectivity, but he had to learn to switch between assignments quickly.

7.1.2 Good Communication Skills

7.1.2.1 Friendliness and Constructive Feedback

During the interviews, all testers stated they were the ones who had to report bugs to the rest of the team; this led to the testers having to adapt friendliness and constructive feedback during their work. All software testers mentioned that the two are essential behaviors. For example, one participant mentioned that *"I was on a project a few years back where I sat next to the developers. When I found a bug, I stood up and walked towards them with a friendly smile"*. Constructive feedback means that the testers focused purely on what went wrong and not whose fault it was, all testers stated that this was important to possess. One tester exemplified: *"Whenever I find a bug, I go to the developer in mind and ask him if it is supposed to be like that. I try not to point any fingers because that is never pleasant for anyone, and it's not appreciated"*. As for non-tester participants, an interaction designer and a developer had the common conception that testers need to be empathetic and positive. They both mentioned that testers should not irritate themselves over the fact that things do not work as supposed to and be positive even during 'dark' times for the team.

7.1.2.2 Meticulous Bug Reports

In addition to being friendly and giving constructive feedback, all testers mentioned that they gave detailed information regarding bug reports. Each one described their procedure as describing what went wrong, the expected result and actual result, and the steps they took to reproduce it. This was particularly observed with Vanessa, which during the interview she stated: *"I'm very heedful (when it comes to reporting bugs), since we have so many services, I usually write which service it applies to, which method, what type of test-data so that they (developers) can reproduce the bug. Then I write expected result, actual result, and the problem itself as an extra sentence"*

Two of the testers liked to be especially diligent when trying to report a bug. One of them said *"...I always try to find whether it is a specific scenario or not. If it's a specific scenario, then I'll attach the user with the specific test-data. If not, then I try to generalize as much as possible like all user's with this type of insurance receive this error..."*. The second tester

said that she also tries to generalize and make certain of the bug's coverage; *"you are supposed to try and reproduce the bug several times with different test-data before you tell and developers. If not, then you've perhaps found a bug that only happens once in a while, and you don't know why..."*

One software tester liked talking to the developers after finding a bug, not for reasons stated above - but rather, to see if it is solvable without spending time logging it on the respective digital task board *"..because it may be so that the bug can be fixed without much, so I don't have to spend much time to register a bug"*. Others have noticed the testers' systematic ways of reporting bugs. One interaction designer said that the tester in her team does things very methodically, *"He is very analytical, he will often find a concrete example of a scenario with the bug, I understand immediately what the fault is and what he's talking about, so he is very communicative"*. Three developers mentioned similar phenomenons exhibited by testers interviewed. They experienced and preferred testers to be as much descriptive as possible.

A developer expressed his frustration for former testing-colleagues not giving enough detailed information. He stated: *"In prior times, the testers just attached a screenshot of something, I couldn't understand where they've clicked, what they've done, what kind of data they've used..."*. However, when prompted the question to another developer, he mentioned that it is context-dependent; *"It varies from project to project. If we have a one-pager of a website, then it does not need to write 'enter the website' because we know that everything is on the site. However, in big, complex systems, details become very important - because of the more details, and easier it is for me to understand where the fault is coming from"*

7.1.2.3 Provide Information, ask for Additional

There have been several occurrences of mentioning that testers need to be good at asking for additional information. A tester mentioned that *"...if the specification is too vague and we find that there can be many different ways to interpret it, then often I'm the one who has to go ask the ones who wrote the specification and find out what exactly do they mean, because I'm the one who specifies concrete requirements to the developers"*. A developer expressed that it was important for testers to ask about commodities and try to find out how things work; *"They are very active in meetings and ask questions about functionalities and what they need to find out about in order to set up test-scenarios"*. Although this was not necessarily observed in Vanessa, she often asked information to Amelia. The reasoning came up during the interview; *"It might have something to do with my personality type, but I don't feel so confident. I feel like I ask stupid questions - and then I'm rather not asking. I'm working with myself to take more space and become confident to ask questions on things I really don't understand..."*. During my period of observation, I saw Vanessa gradually improving

her confidence by becoming more active during meetings, asking questions to customers, and relying a bit less on Amelia.

The majority of the testers mentioned that they communicate, mostly and rigorously, with the developers. Two testers mentioned that they talk mostly with developers because they are the ones requiring additional information while fixing the bugs. The ability to provide more information in an understandable, coherent way was therefore essential. One of the two testers noted that the bug described by 'business' staff are not always very detailed; *"...we have many testers from the business aspect, and it varies how good they are at describing. I often need to add additional information so that the developers don't have to engage in dialogue with them"*. As of the case study, I observed that other workers from the organization Sierra often came to Vanessa for help due to not knowing where to find test-data and the different interoperability between core-systems. In addition, Vanessa often asked Amelia for the root cause of a bug. Amelia spent lots of time investigating the bug before relaying the information to Vanessa.

7.1.2.4 Introversion and Extraversion

One developer noted his perception between the two professions. Whereas developers are more introverts, testers are - at least more than developers - extroverts. This statement was somewhat supported by one of the interaction designers, who said *"if there is a developer that's maybe a bit introverted and not so fond of communicating, then in my mind, that person shouldn't be a tester. Because a tester is supposed to be a middle-man, asking for information and clarifying things and not as much to create and produce than control and discuss - a tester doesn't only sit with his assignment, their assignments come from developers and designers' backlog. A tester should act as a scaffold around the team"*. Yet, it was interesting to note that a different developer commented otherwise: *"The testers are not very full of initiative during meetings and work-hours, I have a feeling that they are very introverted - I think it's because they feel like they're not 'developers' and can't have a say in how things should be done"*. Nevertheless, another interaction designer mentioned something in between; *"He (the tester) is not extrovert, nor an introvert. He is somewhere in between. I know that he enjoys being alone at home or playing board games with his closest friends. However, he is very professional during work and meetings, and you can't notice that he is an introvert. He is very flexible and adapts himself in a situation"*

7.1.2.5 Bridging the Gap Among the Team

Even though the majority of the testers mentioned that they were the middle-man of the team - bridging the gap between mostly interaction designers and developers - one tester explicitly mentioned an example where she was able to bring the teamwork from a minimum to functional: *"When I arrived, the team dynamic was pretty bad. There were almost no communication across professions except for times when it was necessary"*. She men-

tioned how she worked as a middle man, trying to convey the designers' requests and ideas into something more concrete to the developers, and how she managed to convey developers technical jargon to something comprehensible to designers. Over time, each party became more comfortable with her as the middle-man, and she was able to conduct meetings with one from each profession; *"...eventually I got them into the same room, and it went great. They became way more conscious of each other. Designers' idea of only using 15 minutes to implement a single button were not possible, as it turns out the developers will have to make a whole lot of changes, which could result in 20 hours of work just to create a new button"*.

7.1.3 Detail-oriented

In addition to giving detailed information regarding bugs, it became clear that being attentive to detail was important for software testers. Majority of the testers mentioned possessing this trait themselves, and that it is focal to have in their line of work. One tester stated that he possesses a considerable amount of small details on things he's worked on, to the point where he gets dragged into several meetings well beyond his line of profession. In general, the non-tester professionals voiced that having such a trait is determinant for a tester to become successful. Every professional voiced that testers were very detail-oriented and used the trait to improve the product at a great pace. There are mainly two common phenomenons; detail-oriented in the sense of user experience, and in the sense of technicalities.

7.1.3.1 Detail-oriented in Terms of User-experience

Both interaction designers stated that their tester has helped them spot user-experience faults that the respectively overlooked. One of them mentioned: *"I remember he went through an old application, he began to carefully read the text that was there. I thought we were pretty attentive on death insurances - the fact that someone needed to report it when a person dies. He tested the whole process and made me aware of the text; it wasn't pleasant in a highly sensitive situation. So yeah, he is very aware of details"*. The second interaction design mentioned that her colleague has very good knowledge within the domain *"...as a designer, you're also supposed to click through the system and try out stuff. But often, it is the tester that discovers something weird. Like this one time where she came to me and said 'I clicked through this and this, but they are actually the opposite of each other and it shouldn't be possible to click both at the same time (in this domain)'*.

7.1.3.2 Detail-oriented in Terms of Technicalities

Like the designers, the developers perceived that testers need to be detail-oriented. One developer stated that the best testers have this trait without exception, since they notice the 'small-things' that no one else sees. Another developer stated that they are very detailed in terms of logic; *"...this is why I love working with them, because they pick up things that we*

don't see, and things we haven't even thought of. They inform us that 'this is wrong' and 'this wouldn't work because of this method and this module', very logical individuals...". This was similar to the third developer's remark, stating that one of the biggest benefits of testers is that they are able to deduce things the developers haven't thought of, and *"...spot things earlier than us if we were to discover it, and that makes us more prepared."*. Nevertheless, one of the developers stated that it gave them a feeling of psychological safety if a tester is present during development; *"If it's a complex, tangible product, then it is really nice to have a tester next to you. It gives me reassurance in the form of 'okay, he confirms that everything I do is correct. Then it's very likely that what I'm doing is correct"*

7.1.4 Creative

Both designers mentioned that their testers are very creative, and suggests that testers need to be creative. One of them stated that the tester is creative in the sense that he continually tries new combinations on things that could go wrong, mainly because even though tedious and unique faults occur, they can still occur after release. She stated that the tester was also very creative regarding problem-solving, such as suggesting several alternative ways that could resolve obstacles. The second designer stated that her colleague is very much the same, and tries to find unique faults. She recalled one time when the tester noted a problem in the user interface because the tester made up a persona with 52 insurances (this was an application within finance and insurance), a highly unlikely scenario - but still needed in order to ensure that the product is *"idiot-proof"*. Furthermore, she stated that a creative mind is what makes testers valuable; *"...one finds strange things and loopholes by being creative. This is very much appreciated from a tester because we (non-testers) have 'tunneled' ways of testing, there are many non-testers that test precisely how it's supposed to be used, meaning we'd only test the system's behavior when we do things right"*

When it came to the developers, there were mixed opinions on whether testers were behaving creatively. Out of the five developers interviewed, two were quite uncertain whether testers needed to be creative - one of them stated testers should 'think outside the box,' but countered that technical people are quite creative in general. The second developer was unsure if creativity was needed to be able to get into testing. However, he stated that testers might need to be able to find creative test-scenarios to test things out of the ordinary. One developer did not see any correlation between creativity and tester, however stating that neither developer nor testers were known for creativity. The two remaining developers expressed strongly that creativity was a central factor for testers to possess. One of them mentioned that both developer and tester should be creative, but in different ways, *"On the developer side, I reckon it's more how one constructs together things. While for the tester, it's more like 'how can I find ways to destroy the system?"*. The second developer mentioned that being creative was more important for a tester to possess than a developer,

mainly because of the difference in the profession: *"As a developer, you receive a business requirement. Your job is to only fulfill those requirements. You can have a creative process where you construct the architecture, choose frameworks etc. But in the end, you're fulfilling a requirement. Testers are supposed to test a system that's going to work 100%, and there could be so many anomalies. So a creative tester is most likely more important to have in a team, than a creative developer"*

All the tester professionals stated that being creative was necessary for the profession. One participant mentioned the importance of creativity in finding strange bugs; *"I've managed to find weird bugs by being creative, such as mid-way force shutdowns and performing unusual process-sequences. One has to test like that because the users are always creative"*. She also stated that even though one can think logically and see the flow throughout the product by being structured, creativity is needed in order to allow one test the abnormalities during the flow. Another brought-up that this was why being a tester was fun, because of the destructive mindset that entails the effort to try and destroy the system as much as possible and do things in unusual ways that the customer would not have done. A third tester stated that even though creativity can help find weird bugs, it can be so unique that the team does not bother fixing it; *"I daresay I use a lot of creativity to the point that I was told 'the bug you've reported, it is so specific that it only affects one specific customer during a leap year, so we're not going to fix it'*. He also mentioned that it is not always he gets to use his creativeness, mainly due to time pressure. The participant stated that some of the bugs found post-testing could have been found during the QA stage, if only he would have time to think or toyed with it more; *"There isn't always enough time, so I have to focus on getting the most important pieces to work, and then deliver it to the business analysts to test"*

Three of the testers stated that their projects had considerable leeway, both them and the team were highly autonomous to a certain degree. One tester remarked that projects with considerable leeway can allow creativity to flow and enhance ways of testing things. However, she also stated that in order to become creative, one needs to have enough knowledge both from the customer's field and the technical aspect; *"...because if you don't know it, then it'll be hard for you to be creative. You need to understand the domain, know enough about domain-knowledge and technical-knowledge in order to open the doors to creativity"*

7.1.5 Curious

Most of the testers admitted that they are very curious and eagerly to learn when it comes to meeting the unknown. One stated that continually learning new things is his passion. Another commented: *"I think it is super important as a tester if you're eager and possesses a wish to learn. I think acquiring domain-knowledge quickly and uses it...It goes without saying that the more you know, the better you can conduct testing"*. Furthermore, he voiced

that his eagerness has helped somewhat with the customers' relation. He reasoned that it made him ask 'stupid' questions during meetings, such as *"I'm not familiar with that, can you tell me more about it? Is this something we need to test?"*. In this way, he stated that the customer perceived him as committed, which became more accepted towards testers. The tester mentioned that one of his characteristics is that he is a quick-learner; this made him the go-to man for the rest of his team. He is also very inclined to share knowledge to others; *"I focus a lot on learning others my domain-knowledge and testing, and it gets noticed. It's not the fact that they panic when I'm not at work, but rather everyone works much better when I'm here. You contribute in a way to build everyone else up and not making yourself a bottleneck"*

One tester stated that she learned the most through observation and practical experience. She felt that although literature gave her an idea of how things should be done, it became more concrete as she received actual hands-on experience during work. Another tester revealed during the interview that he wanted to start doing security-testing, but was unsure about where to start. His willingness lead him back to becoming a student when he was made aware that the University of Oslo held a practical course regarding ethical hacking. The tester stated that the course helped immensely with deepening his technological knowledge. Furthermore, after the course, he gained pointers on specific areas that he wanted to improve and continually learned new techniques. This similar behavior was also found on Vanessa, who wished to develop skills to be able to perform penetration testing. Throughout my observation, it became known that she willingly took contact with specialists within the subject and received both literature and a pointer on a sandbox website aimed specifically at trying out penetration-testing. She stated that it is very important for testers to continually willing to learn as the software testing theory does not completely reflect the practical world; *"I feel like I'm learning something new all the time, and I have never gotten any training in how to write test-cases either, so I've acquired the knowledge through experience"*. She also mentioned that learning domain-knowledge within the financial sector was challenging, but because the more she got into it, the more intrigued she became, and thus the more curious she was in finding out how the financial aspects worked.

In a similar sense, all testers have or are currently coaching junior testers voiced that junior testers need to be eager to learn new things, and through practical experience if possible. All participants explained that they would show the juniors on how something is done in their own way, and then encourage the junior testers to perform their own testing tasks often through a trial-by-error trajectory. For example, one tester noted *"I usually sit together with them and say: 'okay, here is some test-cases I have written. I can walk you through the first one, feel free to ask any questions', then I'll ask them to write the next test-case. Finally, I'll tell them to write some without my supervision"*. He also stated that this

was a way to challenge themselves: *"I'd ask them: 'is there something you think could go wrong? Are there any other test-cases you can think of? Should we talk to a domain-expert if you're unsure?' In a way to challenge them to a certain degree so that they are used to this way of working"*.

As for non-testers, one developer touched upon the subject. He stated that an excellent tester should always be curious; *"For example, if I say 'we also have to test the APIs.' A good tester would admit that API-testing isn't something familiar, and would request half an hour get more insight in it"*. The developer mentioned that it was always better for a tester to spend some time and come prepared to meetings instead of not knowing what to do even after the information was received.

7.1.6 Structured

As for being structured, professionals voiced that testers should be organized and structured - more so than any other professions in a development team, according to some. During the interviews, all testers disclosed that they are very structured - not only during work hours but also in general. The majority of testers remarked that they usually note everything down to checklists, notes, or in their calendars. One tester stated that she cannot function without structure and that she is not spontaneous at all, and mentioned that writing things down helped her organize and has even managed to catch small details that proved to be significantly important. Another voiced her usual course-of-action when it comes to planning and being structured; *"Every Friday, I look at the calendar to see what's happening next week. Also, once a month, I look through the whole calendar for the next month so that I can plan and leave some room for unexpected meetings"*. A third tester commented that he always write down notes, to-do lists, and checklists, similar findings were also found from both interaction designers and one developer regarding their colleagues. The third tester voiced that even though he is a structured person, he can often be easily distracted and forgetful. The same tester also stated in order to succeed as a tester; one has to be structured; *"...Even if one performs exploratory testing - through just playing around - if you're not structured, then it's possible that you're not able to describe the steps you performed when you have found a bug or even catch some small mistakes. If you're just exploring without being structured, then I don't think you can retrace your steps. So, in my opinion, all testers must be structured"*. During my time of observation, I noticed that Vanessa wrote down a lot of notes and placed each work-activity in her calendar. She also wrote down a lot of notes during each meeting. When prompted, she told me that she wrote about key-points from the meetings and began to see how she could relate the content per her testing work. She told me that she always wrote things down in order to have some guideline or serve as future references that always could prove to be useful.

Most of the developers perceived that testers need to be more structured than them. The common justification is that developers do not have the same responsibilities as testers. One developer said that it is not a crisis for developers if they are not structured, it just means that they may sub-optimally implement the features. At the same time, for testers, it was more critical to check that things work well. Another developer stated: *"For me, it's easy to move some files here and there or write a bunch of code. But for a tester, it is important that absolutely all steps must be reviewed and...testers are supposed to be the devil's advocate and point at details I've overlooked"*. A third developer remarked how the project became much, much more organized and structured when testers arrived. He mentioned that the project lacked both concrete work process and had vague requirement specifications, in addition to lots of things not being documented down; *"...they worked really hard to systematically map, find out and clarify the requirement specifications for us, and even found out things that we never looked at. And as a result, also adjusted and improved our work processes"*

Seeing it from the designers' perspective, one commented that even though she helped with testing - the tests that were conducted by the actual tester are much more thorough because the tester notes down procedures and generates a test-matrix. The second designer mentioned that most testers should like working structured; she mentioned that the tester in her team walkthroughs the product quite precisely, but that the tester has to do it in a structured way to make sure all conditions are covered. Also, the same designer voiced that being a structured tester has benefited the team as a whole; *"...it is easy for us to know with what's going in and out next release, what's tested and not, and it is easy to follow the test-cases she sets up...she always seems to know what the next task is*.

7.1.7 See the Whole Picture

All professionals stated that having to understand the project as a whole is momentous for a tester. The ability to see the total picture with a helicopter-perspective was voiced as a main testers' trait, by those working with them. Additionally, it was found a common pattern among testers, that having such a perspective was somewhat required as a tester, often helping them to find unusual bugs and see things from another perspective.

One interaction designer stated that her colleague has a right mix of knowledge within both domain and technical, there have often been times where the tester have found holes within the journey-experience from consumer-side, which the designer has overlooked; *"...if she only saw my sketches and took for granted this is how insurance works, then it's highly probable that I've missed out on something, so it is always good that she asks: 'is this suppose to work like this'?*. She stated the ability to reflect was necessary for testers, and in order to do that, testers needed to know the domain. At the same time, she remarked that

it would also be hard to report bugs and suggest alternative solutions to the developers if the tester did not have technical competence. She states that her tester is impressive in terms of possessing both knowledge types and that a good tester would require a balance between the two. The second interaction designer expressed similarities with her colleague as well; he is often her go-to man when she has any question regarding domain-knowledge, and: *"...he is the only one in the team who has a good overview of how things are connected, how insurance works, the scope of different insurance types, how things are connected back-end"*. She stated that a common trait with all testers she worked with is that they always had/has a good overview over the service and able to find *"unusual bugs that no one else has control over"*.

As for the developers, all felt testers needed to have the ability to understand the total picture. One developer remarked that testers needed to see the whole, in terms of what the business-side requires and down to the smallest detail from the developer-side. The majority of the developers feel that testers need to have more of the 'complete picture' than developers mostly due to the way these two professions work: they remarked that developers work module-by-module while the testers are supposed to test modules against other modules or the system as a whole; *Developers tend to work in specialized contexts, usually in the form of 'now I'm working on this module.' While a tester is supposed to test this module against all other modules or the service to other services, this makes it more vital for them to understand what response service B and C and D gives. While for developers, it's like 'I got a task to create module A, here you go"*. Similarities were voiced by all other developers, stating that modular development is very common nowadays, and they do not pay as much attention to previous modules as testers. One developer expressed that, according to his prior experience, testers become testers in order to understand the system as a whole and not just the plain technical part. He stated that in projects where no testers were present, he had to take more responsibility in order to have a more totality view on things and thus had to take on more domain-specific tasks.

All tester stated that this trait is important to have in order to make a logical conclusion as to where common bugs can appear and test effectively. One tester indicated that the trait is significant for reasons similar to developers'; that the developers are busy working with the current part and not bothering to look at prior modules, *"...they sit and work with tunnel vision"*, while as a tester the ability to understand things as a whole is a must. She further supported her statement by arguing that testers need to understand the critical parts of the system, and *"It's not about finding most bugs or fewest bugs or run most test-cases. A tester who completed one test-case but have found that the system works the way it is supposed to, is often more effective than a tester who ran a hundred test-cases and have not found anything. So it's about being able to see the total picture"*. Vanessa from the case

study mentioned that this trait was mainly the reason why she chose software tester as her profession; *"I liked the freedom it gave.....I saw how it was during the summer internships. Of course, it varies from place to place, but I saw that as a developer - you receive a task and you do it...They're not very much involved in the process as a whole. I feel it's more exciting to be involved and....have a functional perspective as well, not just the technical"*. Lastly, a third tester, stated that he is constantly curious and able to acquire knowledge at a fast pace, and therefore able to have comprehensive knowledge about totality, more specifically domain-knowledge, than everyone else on the team.

7.2 Hard Skills

In addition to the human factors, hard skills are also perceived by professionals as important in shaping the role of software testers. Hard skills are 'technical' abilities required to perform a specific task within a specific industry. My findings indicate that for software testers, it mainly boils down to two fields: domain-knowledge, and technical-knowledge. Domain-knowledge indicates knowledge of a specific field, such as insurance or loan. Mainly, it is knowledge regarding the customers' field. Technical-knowledge can mean technological knowledge, such as specific testing tools or framework, but mainly it means being able to read and interpret code, as well as having a slight idea on what might at fault in terms of the technical solution.

7.2.1 Domain Knowledge

Most of the developers' perception was that testers needed to have domain-knowledge, more so than other members of the team. The developers felt that currently, the tester in their teams has much better control within the domain, which has helped them much. One developer exemplified an earlier event; *"...we made a system that adhered to the requirement specification, but our tester was proficient within the domain and legal affairs, he came over and told us that the system is wrong. We said that the system gives the correct output, he responded that it is correct - but the problem is that the invoice output is not legally valid. Even though what we had created conformed to the specification, it was invalid in a legal sense."*. A second developer stated that the tester needed to understand more domain aspects than the developers, stating that developers have more requirement on making what the system is supposed to do - but that developers do not always have a clear overview on how it all connects to the domain/business side. In addition, one interaction designer stated that her tester has a much better overview of how the application is supposed to behave under different coverage when someone reports an injury. She said that although she is aware of how the system should respond to simple coverage, the tester is the one who has made her conscious of how different coverage mixed does not produce the right result.

Although most of the testers stated that domain-knowledge is a relevant skill at possessing, there are some distinctive opinions on it. Domain knowledge has its downsides, as one of the testers gets dragged into different meetings because of his high domain-knowledge. He also became very stressed because these meetings took time, and he did not get to complete his work tasks for the day. A second tester notes how she switched domains every six months to two years. She said it was fun to continually learn new domains and that she did not know how much 'behind-the-scenes' different domains had - such as pension, roads, and tunnels. She remarked that this was partly why she chose the profession. A third tester argued that there has to be a balance between domain-knowledge and technical proficiency; *"(Technical) skills are things you need to do your job better - but if you don't have domain-knowledge, then it doesn't matter if you are technically skilled"*. According to him, it is not sufficient to have expertise within testing but also domain-knowledge in order to succeed at doing one's work. It was observed that during a meeting regarding domain-knowledge with Vanessa, Amelia, and three domain-experts, there were unclear definitions on specific terminologies and a somewhat confusion on how processes should proceed (loan, leasing, balancing) between the domain-experts. Both testers later admitted that it was hard for them to follow along when even domain experts disagree on specific topics.

7.2.1.1 Rapid Acquisition

Both interaction designers stated that their testers acquire domain-knowledge at a fast pace. One of them said: *"...he made himself an overview over which coverage/scope we needed to test rather quickly, and why they must be tested, and who should do it, and how far we have come - and he always know these things"*. When asked if there is any direct correlation between acquisition of domain-knowledge and any traits, the same designer said that her colleague is very curious to learning just about everything. One tester stated that rapid acquisition is not necessarily exclusive to testers. She remarked that obtaining domain-knowledge quick is important to professionals regardless of profession: *"If you can't manage that, then it's no point being in this line of work..."*.

7.2.2 Technical Knowledge

In addition to domain knowledge, there have also been mentioned technical proficiency from professionals as an important qualification for a tester to possess. This qualification - according to the interviewees - allowed for an increase in teamwork, specifically between tester and developers, as each can do their job at a more exceptionally.

During the interviews, half the developers began talking about how good their testers were at 'getting their hands dirty' through looking at the code. A common occurrence from the developers was that the testers could give recommendations on alternative solutions by being technically-adept. One developer mentioned: *"...they keep analyzing the code, and they*

can give us advice on how the fault can be solved and attack the issue in a different way than our mindsets". One developer mentioned how it made their job easier through being able to speak the same 'language'; *"...Like when I say 'the code does not compile', they're able to understand what that means, and how long it will take to fix something like that'*.

All the testers acknowledged that possessing good technical proficiency helped them with cooperating with the developers. The most noticeable occurrence of reasoning was that the testers had an understanding of the implementation and thus able to point out what is possible and not possible from a customers' perspective. In addition, testers should be able to give a more detailed description when a bug is found. One tester remarked: *"...not necessarily 'the bug is on line 87', but more 'this bug has something to do with database storage, or this has something to do with module integration,' stuff like that"*. Another tester worked prior as a developer and stated that he appreciated testers with good technical knowledge, and also being able to guess potential bugs before implementation, even suggesting optional ways of implementing features.

One tester noted that she became a tester because she thinks coding is boring; *"I don't have the patience to be a developer... but that working with software - reading, interpreting, and working with codes - is exciting"*. Two testers noted that they are not as technical proficient as they would like to be. Both wanted to develop specialties within test-automation further, and are currently expanding their skill-set. The wish to become more technical was also noted in the case study, where Vanessa told me she wanted to develop more technical qualifications in order to conduct penetration testing.

7.3 External Factors

This section includes factors that indirectly influence the testers. Some testers mentioned factors such as making it enjoyable in being a tester, and others that are needed in order to allow for growth. In this section, I present additional findings that emerged from all interviews and observations. It is also noteworthy to remind the reader that there were no questions allocated to investigate these factors specifically. Instead, these external factors were mentioned by the professionals as a side-track from other questions. Thus, there are few factors here that are mentioned by all participants collectively.

7.3.1 Community of Practice

During the case study, I observed the so-called test status meetings. Since the projects were somewhat intertwined with each other, these testers had a meeting about once every third week in order to talk about any issues and current progress in each respective project. Both Vanessa and Amelia remarked that these meetings were somewhat useful

because of coordination aspects and a place to talk about testing. During one interview with another tester, this topic surfaced - he stated how such meetings were missing from his workplace; *"One thing I've been thinking of is...all the testers from those different teams should have a stand-up once a week, just to exchange challenges, what one is working on, trade experiences"*.

7.3.2 Motivational Factors

The majority of testers mentioned that being able to challenge oneself and learn new things every day was one major factor. Two testers mentioned demanding projects and work-days that are fun, and that it requires them to use their heads. Two others mentioned that they are motivated by learning new things. Additionally, there have been mentions of enjoyable work environments by all testers, which also referred that socializing with co-workers after-work had a considerable impact on their every-day motivation. There have been mentioned several occasions where the team went out to eat, drink, go to the cinema, or even have wine-night in their homes. One tester mentioned that 'work' does not feel like work as much as play and that it helps on teamwork and effectivity; *"...cause then we're more confident in each other and knows that it is easier to ask 'silly' questions"*.

7.3.3 Support System

The support system indicates the ability for testers to seek and receive help, either from the customer's side or from the company's side. One tester expressed frustration in her previous job; *"During the period where I became a tester, I did not have a good leader without that other guy - but he was overworked..."*. Another tester remarked that other members of her team had worked as testers before, so if she needed help, all she could do was to ask them. She said that this 'safety-net' had boosted her confidence in being a tester. Four testers noted that a joint-sharing culture was important within a company. Joint-sharing culture denotes the willingness for one individual to share knowledge with another individual. All four noted that joint-sharing allowed them to learn more efficiently and effectively than if they were alone. In addition, a buddy-mentor system seems to affect the testers' learn-ability. It was observed from the case study that Vanessa always learned new things thanks to Amelia, who acted as a mentor and explained different specifics.

7.3.4 Trust and Respect

Trust was something mentioned by almost all the testers - both trust from the customer's side and internally in the team. One tester mentioned how she gave a strong recommendation not to release an increment of a product, with sound reasoning regarding little time to test. She also made a risk-matrix to show the customer consequences and went through

a list of high-severity faults. The tester stated that the customer decided to hold the release, it was later confirmed that releasing the increment would lead to increased cost from rollbacks and fixing than delaying, and thus the customer became more trustworthy of the tester. Another tester mentioned that the team has a lot of trust in her. She expressed what to prioritize or develop; this was similarly expressed by another tester, stating that because of his domain-knowledge, he rarely gets any objections from both customer-side and internally in the team. He also stated that gaining so much trust allowed him to get involved in many things and learn more. A different tester stated that the more domain knowledge one has, the more external trust one is perceived.

Additionally, one interaction designer mentioned that testers should have a little respect among the team. She exemplified it from one internship where the tester was not sure of the role, and the team did not know what the tester was supposed to contribute with or how correct it is when the tester said something is wrong - which mainly affected the cooperation between the tester and the developer. She stated that the internship thus became a but complication, and even though it was a internship with students, it could apply in the real world as well.

7.4 An Emerging Theory of Software Testers' Human Factors

Since these human factors are not exhibited in isolation, I was able to find correlation between them. This lead to the emerging theory presented as follows. This theory uses framework presented by Sjøberg et al. [73] which consists of four parts: the *constructs* which are the essential elements of the theory, *propositions* which elicits how do the constructs interact with each other, *explanations* on why the propositions are specified, and *scope* in which the theory is applicable. The scope of this theory is software testers working in agile development projects. As for the constructs, they are the themes I have presented in the sections above. I, therefore, will give a description of both explanations and propositions between the constructs. In addition, it was found that some of these factors benefit the team as a whole. As such, I have also included teamwork and noted down the factors that can improve the team's effectivity to work together. It is important to note that the propositions between constructs are entirely grounded from the data. Even though there is a possibility that 'everything could affect everything else' (in a holistic sense), I will only present correlations that have directly been proved with evidence. Thus, the theory and propositions presented here have their roots directly from analysis and findings.

Constructs

C_1 See the whole picture

C_2 Detail-oriented

C_3 Good communication skills

C_4 Structured

C_5 Creative

C_6 Adaptable

C_7 Curious

C_8 Hard skills (domain-knowledge, technical-knowledge)

Propositions

P_1 Hard skills positively affects being able to see the whole picture

P_2 Being detail-oriented positively affects teamwork

P_3 Good communication positively affects teamwork

P_4 Being structured positively affects detail-oriented

P_5 Hard skills positively affects being creative

P_6 Hard skills positively affects being detail-oriented

P_7 Hard skills positively affects being adaptable

P_8 Being curious positively affects hard skills

P_9 Being able to see the whole picture positively affects teamwork

Explanations

E_1 Good domain-knowledge and technical-knowledge are useful for software testers to properly be able to see the whole picture

E_2 Testers with attention to detail can assist both in the designers' and developers' tasks, which increases teamwork with both parties

E_3 Often enough, testers need to speak with every other role in the team. Good communication is therefore vital for testers and can increase team transparency

E_4 Testers who are structured are more likely to be detail-oriented than others, structured testers are a valuable member of the team

E_5 Both domain-knowledge and technical-knowledge will enhance testers creativity - which is used mainly in two dimensions

E_6 Testers should be good at both domain-knowledge and technical-knowledge for them to effectively be detail-oriented

E_7 Domain-knowledge and technical-knowledge should be well versed for software testers, seeing that context-switching between tasks are not unusual in agile teams

E_8 Testers who are curious, are able to attain domain-knowledge and technical-knowledge at an easier and faster pace through their wish to investigate

E_9 Testers who are able to see the whole picture, can reduce others' misunderstandings and forgetfulness, while increasing transparency amongst team members

Scope

S Software testers working in small agile development projects.

Table 14: Constructs, Propositions, Explanations, and Scope of the Theory

Explanation E_1

Testers need to see the whole picture. In other words, what business-side requires, if the user-journey is plausible and whether the technical modules work with each other as intended. Having a helicopter view of the whole product is important as a tester, and the ability to do that is greatly increased with good domain and technical-knowledge. Good domain knowledge is useful for both business and interaction design, while technical knowledge helps with reporting the root cause of bugs and 'speaks the same language' with the developers.

Explanation E_2

Testers who are detail-oriented can usually assist in UX's user-journey with their domain-knowledge. If testers also possess adequate technical-knowledge, they can provide alternative solutions for the developers. Additionally, detail-oriented also tends to lead to meticulous bug reports, which benefits the developers the most. Findings show that being attentive allowed testers to catch mistakes earlier, which had an increase in teamwork, effectiveness, and motivation.

Explanation E_3

It emerged from the interviews that the agile software testers were often needed to provide support for their fellow team members, by clarifying requirements or providing additional information. This often lead to an increase in teamwork through transparency, efficiency, and constructiveness, suggesting that testers have much social navigation to do as to mitigate potential negative team dynamics.

Explanation E_4

Most of the testers stated that writing check-lists and creating notes allowed them to free up their brain and were more likely to be attentive of minor details that no one else had given a second thought. Being structured also helped other team members as it became easy for them to follow his/her work of a tester, while also having the tester systematically keeping a tab on what is going in and out of the next release.

Explanation E_5

It was shown that software testers mainly used creativeness in two ways: to conduct software testing by making up different user personas in different scenarios, and to come up with creative ways of testing the technicalities of the system. Both courses of action can be greatly extended in performance with good knowledge in both domain and technical aspect of the product.

Explanation E_6

Being detail-oriented revolves around two main phenomenons: assisting in shaping the user experience, and the technical part of the product. Each phenomenon requires good proficiency within domain-knowledge and technical-knowledge respectively in order to act as an effective, detail-oriented tester.

Explanation E_7
In an agile environment, software testers need to be adept context-switchers. Having good knowledge in the domain and technical field of the product can greatly enhance the ability to respond quickly to changes, such as changing, adding, or improving test cases, and foster effective communication with domain experts, product owners, or technical staff in order to shape the product to its highest quality.

Explanation E_8
Curious software testers, who possesses a desire to investigate, are able to gain hard skills at an easier and faster pace, be updated on the current scenarios, and thus can conduct testing more properly. It was found that testers who are enthusiastic, willing, and able to actively learn on their own made sure to take up every learning opportunities, and excel themselves in both domain-knowledge and technical-knowledge. Additionally, curious testers liked to learn through practical experience, which often resulted in continuous growth

Explanation E_9
Being able to retain a helicopter view of the whole product makes it easier to resolve issues and potential mistakes coming from the team. Testers who are able to see the whole picture can bridge the gap between the domain and the technical part of the product, while also commenting on each respective part on what to keep in mind for the other during implementation. This has the potential to increase team transparency, making teamwork more fruitful while keeping misunderstandings and forgetfulness to a minimum.

Figure 9 on the next page shows the corresponding illustration of the theory. Factor that have a direct influence on each other are highlighted with arrows. The factors listed with arrows have according to my data, a direct casual relationship to each other and teamwork. In addition, since the human factors are so intertwined and complex to research, is it difficult to rank which ones are the 'best' to possess. These human factors presented in this theory, therefore, adhere to a 'flat-structure,' meaning no trait is better than the other.

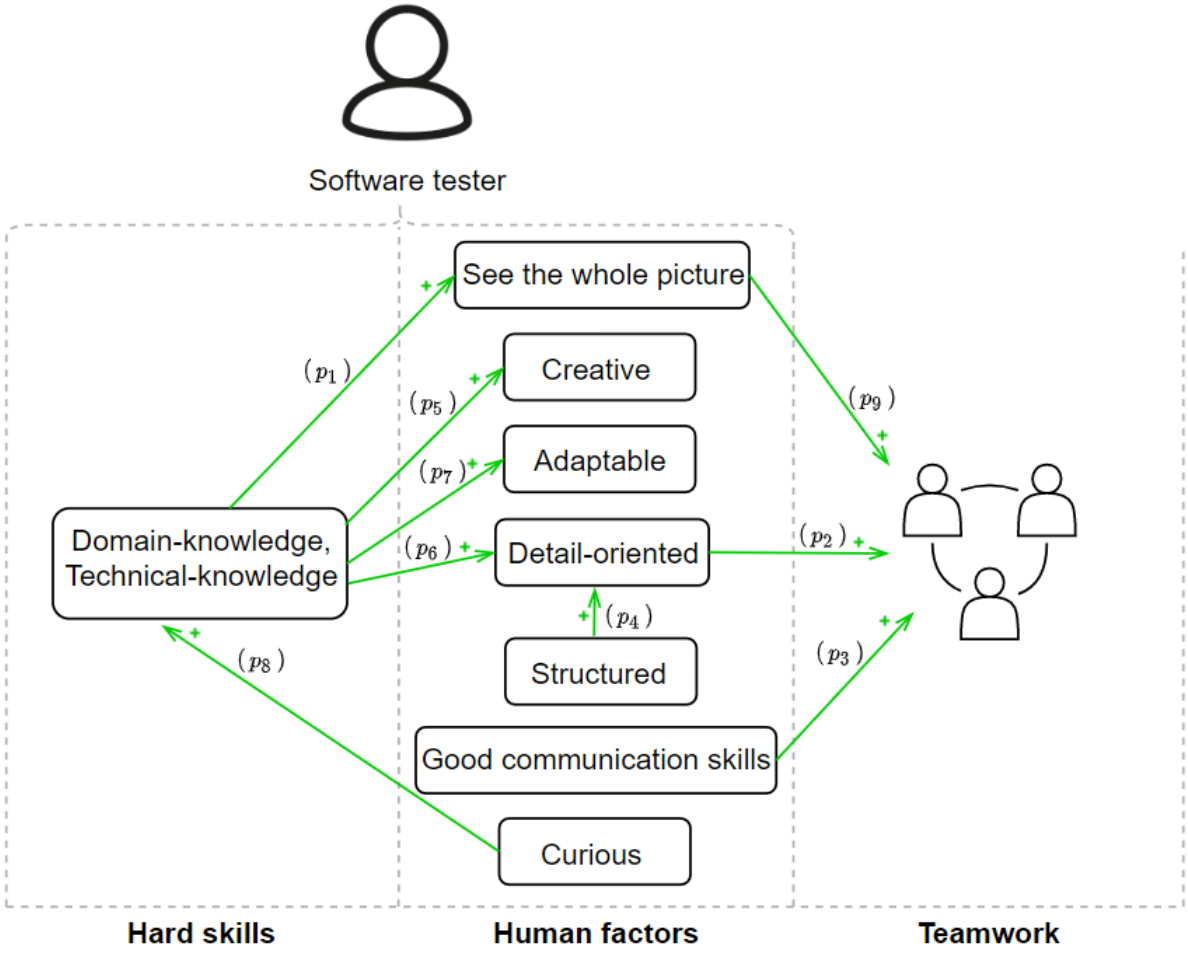

Figure 9: A Theory of Human Factors

8 Discussion

This section discusses the results presented in the previous section. Specifically, they are used to answer the research question proposed in section 1.2 on page 1. I also discuss my findings with existing research. Since my themes are primarily concerning human factors, I compare my results in light of the categories I introduced in section 4.2 on page 19, and discuss whether the results confirm existing results or have found gaps. I then move on to *implications for practice* in order to shed some light on how both relevant organizations and newcomers to software testing can benefit from reading this thesis. Consequently, *implications for theory* detail how other researchers may be able to use my research. Finally, I present the *validity* and the *limitations* of my results.

8.1 Adaptable

There is a great variety of human factors identified in the previous section. Adaptable was identified in conjunction with testers who worked in agile environments. Cavin's aim of study [14] found that military veterans returning to the civilian workforce were suited to be software testers, as many of them had similar characteristics. Communication skills, dedication, adherence to schedules, working under pressure, and commitment - in which some of my findings do align - and adaptability; veterans are trained to think quickly and adapt quickly to any situations. It is interesting to note how many similar traits were found in veterans as in mine. Additionally, Ekwoge et al. [25] mentioned adaptability was needed for new tool usage and techniques of testing. My results suggest that although adaptability does indeed benefit the technical proficiency, the trait is more holistic in the sense that being adaptable also concerns domain-knowledge, and the ability to quickly switch between different context and different mindsets.

8.2 Good Communication Skills

My findings show that having good communication skills was perceived as important for agile software testers, as was also found in earlier studies. Kanji et al. [46] investigated what factors to consider for effective software testers; they found out that dedication, thoroughness, and punctuality are important factors. Additionally, the majority of the participants also mentioned interpersonal skills: the ability to communicate and interact with others. My findings seem to support theirs, as communication skills were found to be quite valuable - if not more important than 'logical' skills. During the interviews with all non-tester participants, when asked if they preferred a tester with good communication skills and sub-par technical skills or vice versa, all participants chose tester with good communication skills. Deak [22] found out that the most frequently cited characteristics for software engineers did not apply to software testers and that good communication skills are the most cited

characteristic for a tester to possess in their results. Consequently, Florea et al. [26] found that the most popular traits asked of a tester were analytical problem-solving skills, communication skills, team-play, and independent-working skills. A recent study showed that people were more careful in their communication if a conflict was thought to occur [33]. As the software testing role implies bringing unwanted news to the team, my findings confirm the need for good communication skills, in a way that does not provoke conflicts.

8.3 Detail-oriented

It was interesting to note that the majority of the professionals voiced that being detail-oriented was imperative for a software tester, since it was useful in both user-experience and technical solutions. This was also in line with the study conducted by Kanij et al. which most respondents agreed this trait was something a good software tester should have [45]. However, Capretz et al. found a set of demotivated factors, in which Cuban software testers were demotivated by the requirement of detail-oriented skills: *"In fact, 'finding mistakes' was the most voted CON by Cuban professionals'* [13].

8.4 Creative

My study shows that being creative was perceived as necessary mainly in two courses of action: to conduct software testing by making up different user personas in different scenarios, and to come up with creative ways of testing the technicalities of the system. Even though creativity was useful for all the roles in software development, the degree of importance varies greatly amongst the roles; Li, Ko, and Begel's findings suggest that creativity and being systematic is a 'should-have' trait amongst software developers [52]. My findings indicate that these traits are 'must-haves' for testers. Even though creativity is useful for all the roles in software development, this comparison can indicate a variation in the degree of importance of the trait amongst the roles. My findings also confirm the results obtained by Itkonen et al. [41], who studied knowledge related to conducting exploratory testing, and found out that this testing approach emphasized/nurtured diverse and creative opportunities of testing. Santos et al. [69] found out that creativity was a significant factor in motivating software testers. Although my research did not strictly focus on this aspect, it is not impossible to think that there could be a relation, as all testers expressed that it was fun to work when creativity was involved. Deak et al. [22] found that although creativity was one of the tester characteristics that was most frequently mentioned amongst interviewees, it was lower ranked than both communication skills and detail-oriented skills, with communication skills deemed most wanted. Although I did not sort my characteristics in ranked order, my discussion regarding communication skills does seem to hint that the characteristics support the findings of Deak et al. Kanij et al. [45] also stated that being creative was an important trait to have. However, unlike Deak et al., they did not sort the

characteristics in ranks.

8.5 Curious

As for curiosity, Kanij et al. found participants agreeing that intellectual curiosity was an important characteristic, implicating that the trait was useful in enhancing the destructive mindset of a tester and helps in deliberately trying to 'break' a system. Their study also revealed that testers should make an active effort to improve their work. The need for learning both new testing techniques and about the *"problem and business domain of their work"* proved to be a common occurrence from their software tester respondents [45]. In a similar sense, Deak et al. found that participants considered being curious and eager *"incentive for continuously improving the understanding of the product"* as well as coming up with unusual testing scenarios [22]. My findings support somewhat both claims, as all professionals I get to interview regarding this factor, said it was an essential factor for a tester to have. My findings also build upon results from Deak et al. in the sense that testers are indeed continuously improving the understanding of the product through continuously learning domain-knowledge and technical-knowledge; although leaning more towards the prior than the latter.

8.6 Structured

Most testers stated that writing check-lists and creating notes allowed them to free up their brain in order to focus on other things. Kanji et al. utilized the Big Five Taxonomy from psychology to highlight testers' personalities. Their findings show that testers generally have a higher level in conscientiousness compared to other roles - conscientiousness being orderliness, self-discipline, hard-working, and dedication [46]. My results show that testers tend to be more organized, detail-oriented, and structured - which all can be subsumed under conscientiousness. However, my findings will also suggest that testers show a high level of openness due to traits such as creativity and curiosity.

8.7 See the Whole Picture

My results show that testers need to have, in addition to technical abilities, an even more complex overview than other roles in agile teams. This finding supports Florea et al. [27] with regards to the number and diversity of skills necessary for the software testers. Additionally, my current results show that testers are leaning more towards domain-knowledge over technical-knowledge - which also somewhat applies to Florea et al. - In their study on what employers look for in software testers' skills, findings show that domain-specific knowledge is less of a focus than actual testing skills when looking for testers to hire, except for area pertaining to financial services software. This can alter the current results in that domain-knowledge is more critical than technical skills, since all the data gathered

were from professionals working within the financial sector. Li, Ko, and Begel conducted a study in order to contribute to holistic, developer-centric insight into what distinguishes great software engineers [52]. Aside from being a competent coder, their research suggests that great engineers should possess ”internal personality traits,” such as the ability to engage with others, decision-making skills, and continuous learning [52]. The majority of my traits were identified in their study. However, the trait of being able to see the total picture was not present during their study - which might hint the trait being more prevalent for testers than developers or team members in general.

Livonen et al. investigated characteristics of high performing testers - characterized by high defect detection rate and seen as important by managers and other testers alike [55]. They found that experience, motivation, personality, and ability to reflect as important. My findings also somewhat support each of these characteristics. Firstly, they referred ability to reflect as the ability to maintain the big picture, which my findings positively support. Secondly, experience relates to both implementation and domain knowledge - my findings show that testers should have a balance of both. This could indirectly enhance the 'devaluation' career perception of testers presented by Capretz et al. and Shah et al. [13, 71]. In their findings, developers state that people become testers because they are not as proficient in technical skills. My results show that testers need to have a more overview - or the ability to see the whole picture - than developers. These rumors likely occur because testers are too busy with the totality and thus, can lean more towards domain-knowledge instead of the technical knowledge. Testers are not necessarily bad regarding technical proficiency, but they have to have less focus on it and more on the domain as they are seen as team members needing to have a helicopter perspective in order to perform their job well.

My findings also support Hernandez et al. [36], who found out that testers felt a positive attitude towards software testing and do not consider it a dull and monotonous activity. This is in line with my findings, suggesting that testing is not an 'unattractive career.' Moreover, Hernandez et al. found out that software testers work on a wide variety of topics, because it *“requires a complete view of the software”*, this is also in line with my finding, suggesting that software testers need a totality view on the whole system as well as communication skill as a necessary trait to possess, to factor for effective collaboration with other departments. In essence, my findings also seem to support this, as testers often work as the middle-man within in the team and have to foster communication between different professions.

8.8 Additional Findings

8.8.1 Software Testing as a Profession

Capretz et al. [13] found out that testing was not a popular profession due to the treatment as the profession as 'second-class citizens' and unattractive career development. Currently, this perception has drastically changed, as all software testers voiced the full recognition of the merits of their role. Furthermore, both the testing and non-testing professionals stated that testers were seen on the same line as other roles and that the teams had a flat-structure. This incidentally improved their motivation in improving their work - which is an exciting find considering existing research such as Shah et al. [71], who found that testing is a stepping stone for the developer career. However, my findings have shown that the majority of the testers wished to become more in-depth in the technical test, which is in line with the prediction of Capretz et al. - stating that the testing profession is: *"changing with the advent of agile methods, DevOps and other paradigms"* [13].

8.8.2 Motivational Factors

Deak et al. highlighted a set of motivational and de-motivational factors [23, 21]. Motivational factors include enjoying challenges, work variety, recognition, and technically challenging work - while de-motivational factors included lack of influence and recognition, unhappy with management, time pressure, boredom, and poor relationships with developers. My findings align with Deak et al. in that all testers loved challenging oneself - both domain and technical - and have work variety. Similarly, Deak et al. defined recognition as *"...awareness of the importance of testing, both among management and development teams, as well as positive feedback received from developers in relation to discovering and fixing bugs"*. This behavior was similarly found out from the interviews and observations, and the testers felt verily appreciated both from the customer and the team internally - which likely had a positive impact on their motivation and the wish to perform their work well. As for the de-motivational factors, they were not visible in my findings. All testers had good relationships with developers, and did not mention boredom. Even though time pressure occurred amongst testers, some worked better under pressure while others did not - but no one mentioned that it was de-motivational. There was one single case from a tester who became somewhat de-motivated from poor management due to a non-existent support system. Deak et al. found out that agile testers were unhappy about their relationship with developers. They stated that it *"might be related to a situation where a company applies a customized version of agile methods"*. I found that testers were generally happy with everyone's relationships. As for the software development process used in projects, it was unsure whether they used a customized version or a combination of Scrum and Kanban.

Livonen et al. also found out that motivation was seen as necessary in order to perform

effective and efficient testing [55] - I showed that testers gained motivation by socializing with co-workers, which can lower the threshold of asking stupid questions and being more friendly with co-workers during work hours. Secondly, Livonen et al. found out that testers were thorough, careful, and systematic - this is both supported from my findings suggesting that testers need to be organized and structured, as well as from existing research previously discussed. Finally, the ability to reflect meant testers' ability to maintain the big picture, allowing testers to focus on important aspects of the software and understand the users' needs. I have also found out that being able to see the whole picture is a desirable trait for testers; it often helped them find unusual bugs and see things from another perspective.

8.9 An Emerging Theory of Software Testers' Human Factors

As for the emerged theory, the existing research regarding human factors of software testers has mentioned some occurrences that can strengthen parts of my causal relations, most of them regarding good communication skills and teamwork. In their study on personality traits of software testers, Kanij et al. found that testers emphasised good communication with developers and customers [45]. Goncalves et al. found that the most indicated difficulty among test professionals was bad communication with developers, and can affect the final quality of the product [32]. Consequently, the study conducted by Shah et al. found that testers were building friendly relations with developers and as a result, the two teams worked more willingly with each other [71]. Their study also found communication gaps among the team created confusion among team members, which lead to missed deadlines. Furthermore, findings from Hernandez et al. shows that the most effective factor for collaboration with others was based on two main human factors: *"openness and attitude of the people and the understanding of shared goals"* [36]. These are all in line with my current theory. However, my theory also suggests that it is not just communication with developers that should be emphasised. Rather, it is important that testers need good communication and relations with interactions designers, developers, project leaders, and product managers alike in order to attain good teamwork. My results also expand the findings from the study conducted by Hernandez et al. [36], in that apart from openness and understanding of shared goals - friendliness and constructiveness are also emphasised for software testers.

Livonen et al. found from their interviews that domain-knowledge is most useful when it is utilized for understanding what the user is trying to achieve with the product, and to understand what parts of the software would be most crucial for testing [55]. Their findings resemble quite to my theory in that domain-knowledge can positively affect creativity, in the sense that it is used in two courses actions; to conduct testing by making up different user personas in different scenarios, and to come up with ways to test the different parts of the system - both courses that seem to quite fit explanations from Livonen et al. However, in

my theory, I have showed that not only is domain-knowledge needed in order to conduct this type of testing creatively, technical-knowledge is also important for understanding things such as interaction between front-end and back-end, APIs, and data-flow between different modules of the system. This, in turn, can allow testers to increase their understanding of how the whole system works, and thus able to creatively create unique user personas that aims to break down the product.

Furthermore, Itkonen et al. found that exploratory testers need both domain-knowledge and system-knowledge. Both knowledge types can be divided into two categories and applied respectively: *"focused knowledge of separate features or details, and holistic knowledge of interactions"* [41]. Their findings presented are very much similar to mine, in the sense that focused knowledge resembles detail-oriented and holistic the ability to see the whole picture. Following this, it can be argued that my findings support their study, in that both domain-knowledge and technical-knowledge can affect the ability to be detail-oriented - exercising focused knowledge - and to see the whole picture - exercising the holistic knowledge of the product.

Findings from Kanij et al. also identified a number of factors that relates to the effectiveness of software testing, with good domain-knowledge being mentioned by respondents as very important [45]. However, some of them also mentioned the ability to learn resulted in continuous growth of knowledge and fostered experience, to the point where testers were able to identify error-prone parts to focus testing on. Results from Matturo showed that one of the top five soft skills that are usually demanded by software companies is eagerness to learn [57], and as mentioned before - findings from Deak [22] shows that curiosity was advantageous for coming up with unusual testing scenarios, as well as an incentive for continuous learning of the product. My theory seems to support these studies, that it is important for software testers to be curious. However, I have shown that they should be curious of not only domain-knowledge, but also technical-knowledge in order to have a better understanding of the product, and able to conduct testing effectively. My findings also show that curious testers possessed an independence to learn new things unprompted by others, strengthening the findings from Deak, in that curiosity is an incentive for continuous improvement.

In addition to the current findings from existing research, my theory proposes new relationships among these human factors and hard skills. I have found that the ability to see the whole picture affects teamwork as a whole. Stated in the results sections, majority of the testers mentioned that they are usually the middle-man of the team. By retaining a helicopter view of the product, testers bridge the gap between domain aspect and the technical aspect by conveying comprehensible information to the other party. Teamwork became

more productive and transparent as testers reminded each party of the knowledge field that they are not well versed in. Even when agile teams foster multi-disciplinary teams, it can still get quite difficult for designers to understand technical jargon spoken by developers. Additionally, the theory shows that being detail-oriented as a software tester can positively affect teamwork. This is shown by software testers able to spot user-experience faults that interaction designers overlooked, as well as technical abnormalities that developers have mistakenly forgotten or ignored. Both of these aspect will affect team effectivity and efficiency. Furthermore, it was shown that testers generated a feeling of psychological safety when they worked with other professions, because it gave other team members reassurance that the software tester confirms every aspect and thus, it is less likely that actual mistakes appear.

My theory also shows that a structured software tester is more detail-oriented. Specifically, structured in the sense that testers are more systematic and organized, and writes down everything in check-lists and to-do's. While some testers note things down to serve as guidelines or saved as future references - acting as a reminder on details that everyone else had forgotten, other testers have noted that writing things down freed up their head to focus on other things, and were thus able to pick up on abnormal details. Nevertheless, techniques such as exploratory testing still requires a somewhat systematic approach to conduct in order for one to know what is been tested and what not, and to retrace their steps when a bug is found. Software testers that are structured, together with their thoroughness and systematic mapping, can thus spot deviations much easier. Finally, my theory indicates that both hard skills: domain-knowledge and technical-knowledge, will affect the ability to be adaptable for an agile software tester. Seeing from both the case study and interviews, software testers' tasks varies greatly between domain-specific to technical-specific. Domain-knowledge is of minimal help during a tasks that aims to test the technical solution and vice versa. Software testers are often helping both designers and developers in their work, which makes it apparent that good knowledge within both domain and technical should be needed in order to adapt to the constant context-switching.

8.10 Implications for Practice

The motivation for writing this thesis was to figure out the desirable human factors for a software tester to have, mainly because I intend to work as one in the near future, but also because I have a fondness over the software testing field. The results of this thesis should, therefore, be considered highly relevant for the industry in the sense that to-be testers can get a sense of the factors presented in this thesis by setting a frame of expectations for the role, and existing software engineers thinking of a career within software testing. This thesis can also benefit the industry by providing a set of relevant skills / traits that testers should possess, useful as a check-list to those in charge of hiring new testing personnel -

for example, giving interviewees a case to work on before the interviews might prove to discover some skills such as being creative, structured, detail-oriented, and able to see the whole picture.

As for organizations with existing testers, these personal skills can prove to be challenging to modify. It is therefore recommended for organizations to apply the factors presented in section 7.3 on page 57. *Community of practice* enabled testers to have a place to discuss issues, share their experience, and receive advice. A community of practice in this sense can mean weekly-meetings for all testers, online forums dedicated to testing, etc. This can benefit immensely for newcomers to testing, as a community of practice can act as scaffolds for supporting growth. Meetings could be done either online or offline, but from what I have discovered - offline works best as individuals are physically there. But if meeting present proved to be challenging to manage, online with video call works just as well - according to the findings. Secondly, ensure that testers are continuously learning and working on challenging projects - but it also should be the right amount of challenge. Organizations should be in close contact with testers regarding this aspect. Finally, co-workers should aim to socialize with each other during off-work hours. It was shown that motivation increased when testers got to know everyone in a social setting, which made it easier to coordinate with each other during work.

Organizations could also incorporate a support system for testers. In addition to constructing Community of Practice, there is a need for joint-sharing culture. Cultural values that highlights the 'sharing-is-caring' mentality and that sharing is beneficial for everyone. Testers should not feel to forced 'give-away' their knowledge that makes them unique. Instead, testers need to be automatically willing to share their knowledge to impact everyone else positively. Even with a support system in place, testers could still have trouble asking for specific advice within their specific work context. Buddy-mentor relationships should, therefore, also be established within the team as guidance for any newcomers learning the ropes. As for trust and inter-team relationships, testers need to be friendly and respect the customer and other team members' decisions. Seeing as trust comes from experience, buddy-mentor relationships further enhances its importance as to gain the customers' trust and add some credibility. For a team to coordinate well, everyone should respect everyone else. Therefore, the suggestions for implementing these external factors also apply to other professions working in a software development team.

8.11 Implications for Theory

For researchers looking to build upon the theory I presented in 7.4 on page 59, it is important to remember that these human factors are not set in stone. Researchers may also reveal other traits that are not present in this thesis or find different correlations compared

to my theory. It is important to remember that the results were found in a specific group of professionals operating in a specific area, working for a specific organization in a specific geographical location. Nevertheless, the theory I presented had a direct connection in my data, meaning some other relations could have been omitted due to the lack of concrete data proving the causal relation. Researchers could look deeper into cultural variations and how that impacts the factors that are needed in order to make an effective software tester. Since many software development organizations are multinational and multicultural, researchers could differentiate the findings discovered here versus factors needed in order to cross-cultural boundaries. The human factors described within this thesis have emerged from individuals working in agile frameworks (Scrum, Kanban), and thus the newly-founded theory is mostly applicable to this context.

8.12 Validity

In this section I describe the steps I took in ensuring the four validity types presented in section 5.5 on page 34 - *Construct validity*, *Internal validity*, *External validity*, and *Reliability*.

8.12.1 Construct Validity

Construct validity was supported by using different data-gathering methods. I also placed interview data as a higher priority than observational data to reduce bias or subjectivity. I did not twist the data to fit my personal view or opinion, and I refrained from using any data that was open to more than one interpretation as evidence. Nevertheless, I followed Yin's principle [82] by maintaining a chain of evidence throughout this whole study.

8.12.2 Internal Validity

Since internal validity elicits the extend to which pieces of evidence support the causal relationship between cause and effect, I upheld this validity type through using multiple sources of evidence before claiming a theme - such as observation notes, interview transcripts, and meeting notes.

8.12.3 External Validity

The aspect of generalizability is hard to impose within the field of Software Engineering. This is due mainly to the vast variations in organizations, projects, teams, and processes, etc. One of the countermeasures I applied was to interview individuals that were working in different contexts - but that all used agile processes - in order to try and generalize the findings in a vacuum. A much better way to reduce external validity was to conduct a multiple-case study. That, however, proved to be challenging to attain.

8.12.4 Reliability

Reliability is important to ensure that future researchers can reproduce the results. This can be achieved by having multiple sources of evidence. I enforced this aspect by using quotes, observational field notes, and conversation logs in my results chapter. Additionally, I created a Case Study Database used during data analysis. Both principles are recommended by Yin [82], and hopefully, future researchers can replicate my method.

8.13 Limitations

It is essential to note the limitations of this study. The most significant limitations of my thesis are context, time, and inexperience. This thesis was conducted mainly using data gathered from professionals working work at one specific company in one specific geographical location. The findings discovered can, therefore, be influenced by both cultural and organizational values/standards. Had this been a multi-case study with different organizations across borders, the human factors discovered might have been dissimilar than my current findings. Secondly, if there were more time, I would have been able to confirm my findings more securely and most likely fill some gaps and found discrepancies within my results by gathering more data. Consequently, my inexperience in research has, in all likelihood, impacted this study. However, I have tried to minimize this impediment through reading as much as I could on the different research artifacts, and followed as closely as I could on the research methods chosen. I have kept my own bias out of the study and had a systematic, organized mindset when conducting this whole study.

8.13.1 Limitations Regarding the Data

There are a few limitations to the current study, given its qualitative nature. It is worth mentioning that interviews have several disadvantages. For example, poorly constructed questions may lead to poorly given answers - I tried, to the best of my knowledge, to construct concrete questions, so that interviewees understood beyond doubt the questions. Secondly, if the interviewees do not recall an event correctly, their answers may vary depending on their own interpretation at the time or after they have given some thoughts. This is especially important to note for studies researching interpersonal aspects. I interpreted this as a significant data analysis and interpretation validity, as it was not possible to confirm the interviewees recall of the events. Thirdly, the interviewees might have given erroneous answers, which may have further increased my own bias. Neutral non-leading question formulations were, therefore, crucial for me to construct and consider. Measures were taken prior, and during these interviews, such as discussing with my supervisor and continually refining the interview-guide. I also endeavored to minimize my own bias through conducting multiple interviews with multiple professionals that had no work relations with each other in order to reduce bias and try to get a generalized sense. Yet, that proved to act like

a double-edged sword - which I explain why in the following text.

Since my primary data was gathered from interviews - half of the interviewees were asked about their own personal perception of testers and their human factors. This might be limited in scope, due to two major reasons. First, the interviewees may not have observed actions that exhibit a specific trait of a tester. Meaning that two different developers working on separate projects may have had different experiences regarding the tester's traits. In essence, one may have observed actions done by the tester that may consider creative, while another has not observed such actions in his/her projects. Secondly, the perception of whether a trait is exhibited is influenced by personal views and opinions. For instance, a developer can describe a software tester by their actions, but however one perceives those actions is subjective: if two developers are describing the same tester, one may state that the tester is creative while the other might not. One's world view directly influences things we do or perceive in some way, and both of these reasons - in my opinion - share a weak spot to the reliability of the data.

9 Conclusion

To sum-up the whole thesis, I conducted a Grounded Theory study consisting of interviews and project observations of an agile software development team within the financial sector, in the period November 2019 - January 2020. The scope of my thesis was to investigate the human factors and skills perceived by the software professionals as determinant for the software tester role.

I presented background information related to the thesis, followed by the conduction of two systematic literature reviews, in order to obtain the overview of the current state of research within the topic I focused on. I described research methods and the reasoning behind why they were chosen, as well as the research context, data collection, and data analysis procedures - all which acted as the foundation of my thesis and theory. The emergent theory is meant to outline the relationship between how the human factors of the role can influence one another, and although the theory itself is open to further development, my findings serve as building blocks for other researchers to build upon. I presented my findings and discussed these in light of relevant research literature, and elicited implications for practice and for theory. Finally, I reflected on the validity and limitations of the study.

My findings show that all the professionals, testers and non-testers, see the following human factors as pivotal for the agile software testing role: able to see the whole picture, having good communication skills, being detailed-oriented, structured, creative, curious, and adaptable. In addition, software testers are also expected to be proficient in hard skills, such as domain-knowledge and technical knowledge. The emerged theory I presented, shows that for software testers, being curious will help in learning domain-knowledge and technical-knowledge. In addition, both domain-knowledge and technical-knowledge will have a influence on being creative, adaptable, detail-oriented, and being able to see the whole picture. Software testers that are structure will positively impact their ability to be detail-oriented. Finally, good communication skills, being detail-oriented, and able to see the whole picture as a software tester will increase teamwork.

In my thesis, I build upon the existing research done on human factors of software testers working in agile software development, contributing to identifying specific human factors for testers, through observation and interviews of both software testers and non-software testers. I constructed an emergent theory, however, as this thesis is inter-disciplinary, it could benefit from research in the psychology field. Future work could arise, such as finding out differences of human factors needed in agile versus sequential development models or explore deeper the traits required for testers and other roles within a development team.

10 Bibliography

[1] M. O. Ahmad, J. Markkula, and M. Oivo. "Kanban in software development: A systematic literature review". In: *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*. Sept. 2013, pp. 9–16. DOI: 10.1109/SEAA.2013.28.

[2] A. Ahmed et al. "Agile software development: Impact on productivity and quality". In: *2010 IEEE International Conference on Management of Innovation Technology*. June 2010, pp. 287–291. DOI: 10.1109/ICMIT.2010.5492703.

[3] R. Vijay Anand and M Dinakaran. "Popular agile methods in software development: Review and analysis". en. In: *International Journal of Scientific and Technical Advancements* 2.4 (2016), pp. 147–150. ISSN: 2454-1532.

[4] Mundita Awotar and Roopesh Kevin Sungkur. "Optimization of Software Testing". In: *Procedia Computer Science*. International Conference on Computational Intelligence and Data Science 132 (Jan. 2018), pp. 1804–1814. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.05.142.

[5] Omar Badreddin. "Thematic Review and Analysis of Grounded Theory Application in Software Engineering". en. In: *Advances in Software Engineering* 2013 (2013), pp. 1–9. ISSN: 1687-8655, 1687-8663. DOI: 10.1155/2013/468021.

[6] Aleksander Bai, Heidi Mork, and Viktoria Stray. "A cost-benefit analysis of accessibility testing in agile software development: results from a multiple case study". en. In: *International Journal on Advances in Software* 10.1, 2 (2017), pp. 96–107.

[7] Nada Bajnaid, Rachid Benlamri, and Boris Cogan. "An SQA e-Learning System for Agile Software Development". en. In: *Networked Digital Technologies*. Ed. by Rachid Benlamri. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2012, pp. 69–83. ISBN: 978-3-642-30567-2.

[8] Rex Black, Erik van Veenendaal, and Dorothy Graham. *Foundations of Software Testing ISTQB Certification*. 3rd. Cengage Learning EMEA; 3 edition, 2012. ISBN: 978-1-4080-4405-6.

[9] Virginia Braun and Victoria Clarke. *Successfull Qualitatie Research: A Practical Guide for Beginners*. en-US. SAGE Publications, 2013. ISBN: 978-1-84787-582-2.

[10] Virginia Braun and Victoria Clarke. "Using thematic analysis in psychology". en. In: *Qualitative Research in Psychology* 3.2 (Jan. 2006), pp. 77–101. ISSN: 1478-0887, 1478-0895. DOI: 10.1191/1478088706qp063oa.

[11] Virginia Braun et al. "Thematic Analysis". en. In: *Handbook of Research Methods in Health Social Sciences*. Ed. by Pranee Liamputtong. Singapore: Springer, 2019, pp. 843–860. ISBN: 978-981-10-5251-4. DOI: 10.1007/978-981-10-5251-4_103.

[12] Amadeu Silveira Campanelli and Fernando Silva Parreiras. "Agile methods tailoring – A systematic literature review". In: *Journal of Systems and Software* 110 (Dec. 2015), pp. 85–100. ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.08.035.

[13] L. F. Capretz et al. “Studies on the Software Testing Profession”. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. May 2019, pp. 262–263. DOI: 10.1109/ICSE-Companion.2019.00105.

[14] Jerry D. Cavin. “The Role of Human Factors in Veteran SQA Training”. In: *Procedia Manufacturing*. 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015 3 (Jan. 2015), pp. 1535–1542. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2015.07.416.

[15] H. Frank Cervone. “Understanding agile project management methods using Scrum”. en. In: *OCLC Systems & Services: International digital library perspectives* (Feb. 2011). ISSN: 1065-075X. DOI: 10.1108/10650751111106528.

[16] Kathy Charmaz. *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. SAGE Publications, 2006. ISBN: 978-0-7619-7353-9.

[17] J. Chóliz, J. Vilas, and J. Moreira. “Independent Security Testing on Agile Software Development: A Case Study in a Software Company”. In: *2015 10th International Conference on Availability, Reliability and Security*. Aug. 2015, pp. 522–531. DOI: 10.1109/ARES.2015.79.

[18] VersionOne CollabNet. *12th Annual State of Agile Report*. en-US. Tech. rep. 12.

[19] Juliet Corbin and Anselm Strauss. “Grounded Theory Methodology: An Overview”. In: *Handbook of qualitative research*. 1994, pp. 273–285.

[20] Benjamin Crabtree and William Miller. *Doing Qualitative Research*. SAGE Publications, 1999.

[21] Anca Deak. “A Comparative Study of Testers’ Motivation in Traditional and Agile Software Development”. en. In: *Product-Focused Software Process Improvement*. Ed. by Andreas Jedlitschka et al. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 1–16. ISBN: 978-3-319-13835-0.

[22] Anca Deak. “What Characterizes a Good Software Tester? – A Survey in Four Norwegian Companies”. en. In: *Testing Software and Systems*. Ed. by Mercedes G. Merayo and Edgardo Montes de Oca. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 161–172. ISBN: 978-3-662-44857-1.

[23] Anca Deak, Tor Stålhane, and Guttorm Sindre. “Challenges and strategies for motivating software testing personnel”. In: *Information and Software Technology* 73 (May 2016), pp. 1–15. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2016.01.002.

[24] Saru Dhir and Deepak Kumar. “Automation Software Testing on Web-Based Application”. en. In: *Software Engineering*. Ed. by M. N. Hoda et al. Advances in Intelligent Systems and Computing. Springer Singapore, 2019, pp. 691–698. ISBN: 978-981-10-8848-3.

[25] O. M. Ekwoge, A. Fontão, and A. C. Dias-Neto. “Tester Experience: Concept, Issues and Definition”. In: *2017 IEEE 41st Annual Computer Software and Applications Conference*. Vol. 1. July 2017, pp. 208–213. DOI: 10.1109/COMPSAC.2017.232.

[26] Raluca Florea and Viktoria Stray. “Software Tester, We Want to Hire You! an Analysis of the Demand for Soft Skills”. en. In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Juan Garbajosa, Xiaofeng Wang, and Ademar Aguiar. Lecture Notes in Business Information Processing. Springer International Publishing, 2018, pp. 54–67. ISBN: 978-3-319-91602-6.

[27] Raluca Florea and Viktoria Stray. “The skills that employers look for in software testers”. en. In: *Software Quality Journal* 27.4 (Dec. 2019), pp. 1449–1479. ISSN: 1573-1367. DOI: 10.1007/s11219-019-09462-5.

[28] Carlos Alberto Fortunato et al. “Quality Assurance in Agile Software Development: A Systematic Review”. en. In: *Agile Methods*. Ed. by Tiago Silva da Silva et al. Communications in Computer and Information Science. Springer International Publishing, 2017, pp. 142–148. ISBN: 978-3-319-55907-0.

[29] Cesar Gil et al. “Agile Testing Practices in Software Quality: State of The Art Review”. en. In: *Journal of Theoretical and Applied Information Technology* 92 (2016), p. 9.

[30] Lisa Given. *The SAGE Encyclopedia of Qualitative Research Methods*. Vol. 1 & 2. 2008. ISBN: 978-1-4129-4163-1.

[31] Barney G. Glaser and Anselm Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. 1967. ISBN: 0-202-30260-1.

[32] W. F. Gonçalves et al. “The influence of human factors on the software testing process: The impact of these factors on the software testing process”. In: *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*. June 2017, pp. 1–6. DOI: 10.23919/CISTI.2017.7975873.

[33] Lucas Gren. “The Links Between Agile Practices, Interpersonal Conflict, and Perceived Productivity”. In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. EASE’17. event-place: Karlskrona, Sweden. New York, NY, USA: ACM, 2017, pp. 292–297. ISBN: 978-1-4503-4804-1. DOI: 10.1145/3084226.3084269.

[34] Geir K. Hanssen et al. “Quality Assurance in Scrum Applied to Safety Critical Software”. en. In: *Agile Processes, in Software Engineering, and Extreme Programming*. Ed. by Helen Sharp and Tracy Hall. Lecture Notes in Business Information Processing. Springer International Publishing, 2016, pp. 92–103. ISBN: 978-3-319-33515-5.

[35] T. D. Hellmann, A. Hosseini-Khayat, and F. Maurer. “Supporting Test-Driven Development of Graphical User Interfaces Using Agile Interaction Design”. In: *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. Apr. 2010, pp. 444–447. DOI: 10.1109/ICSTW.2010.35.

[36] T. P. R. y Hernández and N. Marsden. “Understanding software testers in the automotive industry a mixed-method case study”. In: *2014 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*. Aug. 2014, pp. 305–314.

[37] Rashina Hoda, James Noble, and Stuart Marshall. “Grounded theory for geeks”. en. In: *Proceedings of the 18th Conference on Pattern Languages of Programs - PLoP ’11*. Portland, Oregon: ACM Press, 2011, pp. 1–17. ISBN: 978-1-4503-1283-7. DOI: 10.1145/2578903.2579162.

[38] Marie C. Hoepfl. “Choosing Qualitative Research: A Primer for Technology Education Researchers”. en. In: *Journal of Technology Education* 9.1 (Sept. 1997), p. 16. ISSN: 1045-1064. DOI: 10.21061/jte.v9i1.a.4.

[39] G. Hongying and Y. Cheng. “A customizable agile software Quality Assurance model”. In: *The 5th International Conference on New Trends in Information Science and Service Science*. Vol. 2. Oct. 2011, pp. 382–387.

[40] Liang Huang and Mike Holcombe. “Empirical investigation towards the effectiveness of Test First programming”. In: *Information and Software Technology*. Special Section - Most Cited Articles in 2002 and Regular Research Papers 51.1 (Jan. 2009), pp. 182–194. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.03.007.

[41] J. Itkonen, M. V. Mäntylä, and C. Lassenius. “The Role of the Tester’s Knowledge in Exploratory Software Testing”. In: *IEEE Transactions on Software Engineering* 39.5 (May 2013), pp. 707–724. ISSN: 0098-5589. DOI: 10.1109/TSE.2012.55.

[42] A. Janus et al. “The 3C approach for Agile Quality Assurance”. In: *2012 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM)*. June 2012, pp. 9–13. DOI: 10.1109/WETSoM.2012.6226998.

[43] D. Janzen and H. Saiedian. “Test-driven development concepts, taxonomy, and future direction”. In: *Computer* 38.9 (Sept. 2005), pp. 43–50. ISSN: 0018-9162. DOI: 10.1109/MC.2005.314.

[44] P. Kandil, S. Moussa, and N. Badr. “A methodology for regression testing reduction and prioritization of agile releases”. In: *2015 5th International Conference on Information Communication Technology and Accessibility (ICTA)*. Dec. 2015, pp. 1–6. DOI: 10.1109/ICTA.2015.7426903.

[45] T. Kanij, R. Merkel, and J. Grundy. “A Preliminary Survey of Factors Affecting Software Testers”. In: *2014 23rd Australian Software Engineering Conference*. Apr. 2014, pp. 180–189. DOI: 10.1109/ASWEC.2014.32.

[46] T. Kanij, R. Merkel, and J. Grundy. “An Empirical Investigation of Personality Traits of Software Testers”. In: *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. May 2015, pp. 1–7. DOI: 10.1109/CHASE.2015.7.

[47] Imrul Kayes. “Agile Testing: Introducing PRAT As a Metric of Testing Quality in Scrum”. In: *SIGSOFT Softw. Eng. Notes* 36.2 (May 2011), pp. 1–5. ISSN: 0163-5948. DOI: 10.1145/1943371.1943384.

[48] Vesa Kettunen et al. “A Study on Agility and Testing Processes in Software Organizations”. In: *Proceedings of the 19th International Symposium on Software Testing and Analysis*. ISSTA ’10. event-place: Trento, Italy. New York, NY, USA: ACM, 2010, pp. 231–240. ISBN: 978-1-60558-823-0. DOI: 10.1145/1831708.1831737.

[49] B. Kitchenham and S. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007.

[50] Corey Ladas. *Scrumban - Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi lean series. Modus Cooperandi Press, 2009. ISBN: 978-0-578-00214-9.

[51] Craig Larman and Bas Vodde. *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Addison Wesley, 2008. ISBN: 0-321-61715-0.

[52] Paul Luo Li, Amy J. Ko, and Andrew Begel. “What distinguishes great software engineers?” en. In: *Empirical Software Engineering* 25.1 (Jan. 2020), pp. 322–352. ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-019-09773-y.

[53] O. Liechti, J. Pasquier, and R. Reis. “Supporting Agile Teams with a Test Analytics Platform: A Case Study”. In: *2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST)*. May 2017, pp. 9–15. DOI: 10.1109/AST.2017.3.

[54] Mikael Lindvall et al. “Agile Metamorphic Model-based Testing”. In: *Proceedings of the 1st International Workshop on Metamorphic Testing*. MET ’16. event-place: Austin, Texas. New York, NY, USA: ACM, 2016, pp. 26–32. ISBN: 978-1-4503-4163-9. DOI: 10.1145/2896971.2896979.

[55] Joonas Livonen, Mika Mantyla, and J. Itkonen. “Characteristics of high performing testers: a case study.” en. In: *Proceedings of the International Symposium on Empirical Software Engineering and Measurement, ESEM 2010*. Bolzano, Italy, Jan. 2010.

[56] Michael Lynch, Thomas Cerqueus, and Christina Thorpe. “Testing a Cloud Application: IBM SmartCloud Inotes: Methodologies and Tools”. In: *Proceedings of the 2013 International Workshop on Testing the Cloud*. TTC 2013. event-place: Lugano, Switzerland. New York, NY, USA: ACM, 2013, pp. 13–17. ISBN: 978-1-4503-2162-4. DOI: 10.1145/2489295.2489299.

[57] G. Matturro. “Soft skills in software engineering: A study of its demand by software companies in Uruguay”. In: *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. May 2013, pp. 133–136. DOI: 10.1109/CHASE.2013.6614749.

[58] M. Meyer. “Continuous Integration and Its Tools”. In: *IEEE Software* 31.3 (May 2014), pp. 14–16. ISSN: 0740-7459. DOI: 10.1109/MS.2014.58.

[59] Isadore Newman and Carolyn Benz. *Qualitative-quantitative research methodology: Exploring the interactive continuum*. Southern Illinois University Press, 1998. ISBN: 978-0-8093-2150-6.

[60] Matjaž Pančur and Mojca Ciglarič. “Impact of test-driven development on productivity, code and tests: A controlled experiment”. In: *Information and Software Technology*. Special Section: Best papers from the APSEC 53.6 (June 2011), pp. 557–573. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2011.02.002.

[61] Lucas Paruch, Viktoria Stray, and Charlotte Bech Blindheim. “Characteristic Traits of Software Testers”. In: *Proceedings of the Evaluation and Assessment in Software Engineering*. EASE ’20. Trondheim, Norway: Association for Computing Machinery, Apr. 2020, pp. 371–372. ISBN: 978-1-4503-7731-7. DOI: 10.1145/3383219.3383270.

[62] L. Prechelt, H. Schmeisky, and F. Zieris. “Quality Experience: A Grounded Theory of Successful Agile Projects without Dedicated Testers”. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. May 2016, pp. 1017–1027. DOI: 10.1145/2884781.2884789.

[63] P. Rahayu et al. “Applying usability testing to improving Scrum methodology in develop assistant information system”. In: *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*. Oct. 2016, pp. 1–6. DOI: 10.1109/ICITSI.2016.7858222.

[64] Jane Ritchie and Jane Lewis. *Qualitative Research Practice: A Guide for Sosial Science Students and Researchers*. SAGE Publications, 2003. ISBN: 0 7619 7110 6.

[65] Colin Robson and Kieran McCartan. *Real World Research*. 4th ed. 2011. ISBN: 978-1-119-14485-4.

[66] Johnny Saldana. *The Coding Manual for Qualitative Researchers*. SAGE Publications, 2012. ISBN: 1-4462-7142-0.

[67] Iflaah Salman, Burak Turhan, and Sira Vegas. “A controlled experiment on time pressure and confirmation bias in functional software testing”. en. In: *Empirical Software Engineering* 24.4 (Aug. 2019), pp. 1727–1761. ISSN: 1573-7616. DOI: 10.1007/s10664-018-9668-8.

[68] A. M. dos Santos et al. “Testing in an agile product development environment: An industry experience report”. In: *2011 12th Latin American Test Workshop (LATW)*. Mar. 2011, pp. 1–6. DOI: 10.1109/LATW.2011.5985897.

[69] R. E. d S. Santos et al. “Would You Like to Motivate Software Testers? Ask Them How”. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Nov. 2017, pp. 95–104. DOI: 10.1109/ESEM.2017.16.

[70] James T. Sawyer and David M. Brann. “How to Test Your Models More Effectively: Applying Agile and Automated Techniques to Simulation Testing”. In: *Winter Simulation Conference*. WSC ’09. event-place: Austin, Texas. Winter Simulation Conference, 2009, pp. 968–978. ISBN: 978-1-4244-5771-7.

[71] Hina Shah and Mary Jean Harrold. “Studying Human and Social Aspects of Testing in a Service-based Software Company: Case Study”. In: *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE ’10.

event-place: Cape Town, South Africa. New York, NY, USA: ACM, 2010, pp. 102–108. ISBN: 978-1-60558-966-4. DOI: 10.1145/1833310.1833327.

[72] F. S. Silva et al. “A Reference Model for Agile Quality Assurance: Combining Agile Methodologies and Maturity Models”. In: *2014 9th International Conference on the Quality of Information and Communications Technology*. Sept. 2014, pp. 139–144. DOI: 10.1109/QUATIC.2014.25.

[73] Dag I. K. Sjøberg et al. “Building Theories in Software Engineering”. en. In: *Guide to Advanced Empirical Software Engineering*. Ed. by Forrest Shull, Janice Singer, and Dag I. K. Sjøberg. London: Springer London, 2008, pp. 312–336. ISBN: 978-1-84800-043-8. DOI: 10.1007/978-1-84800-044-5_12.

[74] Anselm Strauss. *Qualitative analysis for social scientists*. Cambridge University Press, 1987. ISBN: 0-521-33806-9.

[75] V. G. Stray, N. B. Moe, and A. Aurum. “Investigating Daily Team Meetings in Agile Software Projects”. In: *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. Sept. 2012, pp. 274–281. DOI: 10.1109/SEAA.2012.16.

[76] Viktoria Stray, Nils Brede Moe, and Dag I.K. Sjoberg. “Daily Stand-Up Meetings: Start Breaking the Rules”. In: *IEEE Software* 37.3 (May 2020). Conference Name: IEEE Software, pp. 70–77. ISSN: 1937-4194. DOI: 10.1109/MS.2018.2875988.

[77] Saurabh Tiwari and Atul Gupta. “An Approach of Generating Test Requirements for Agile Software Development”. In: *Proceedings of the 8th India Software Engineering Conference*. ISEC ’15. event-place: Bangalore, India. New York, NY, USA: ACM, 2015, pp. 186–195. ISBN: 978-1-4503-3432-7. DOI: 10.1145/2723742.2723761.

[78] R. Tommy et al. “Dynamic quality control in agile methodology for improving the quality”. In: *2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)*. Nov. 2015, pp. 233–236. DOI: 10.1109/CGVIS.2015.7449927.

[79] X. Wang. “The Combination of Agile and Lean in Software Development: An Experience Report Analysis”. In: *2011 Agile Conference*. Aug. 2011, pp. 1–9. DOI: 10.1109/AGILE.2011.36.

[80] L. Williams and A. Cockburn. “Agile software development: it’s about feedback and change”. In: *Computer* 36.6 (June 2003), pp. 39–43. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1204373.

[81] Carla Willig and Wendy Stainton Rogers. *The SAGE Handbook of Qualitative Research in Psychology*. 2nd ed. SAGE Publications, 2017. ISBN: 1-5264-2286-7.

[82] Robert Yin. *Case Study Research and Applications: Design and Methods*. 6th ed. Thousand Oaks, California: SAGE Publications, Sept. 2017. ISBN: 978-1-5063-3616-9.

A NSD Consent form

Are you interested in taking part in the research project "Software testing in agile development"?

This is an inquiry about participation in a research project where the main purpose is to analyze human factors involved in a tester occupation. In this letter we will give you information about the purpose of the project and what your participation will involve.

Purpose of the project
The purpose of the master's thesis is to investigate the tester-role working in a project where agile methodology is utilized. The scope of the study includes interviews and observations. The project is conducted in relation to a master's thesis.

Who is responsible for the research project?
University of Oslo is the institution responsible for the project.

Why are you being asked to participate?
You are chosen to participate in this study because you are a person of interest, who works in a part of an agile team.

What does participation involve for you?
Your participation in the project will entail being observed on various occasions, as well as making yourself eligible for interviews. An interview will take approximately 45 minutes and audio recording will be utilized after consent. The questions that will be asked to revolve around thought on your personal traits as a software tester.

Participation is voluntary
Participation in the project is voluntary. If you choose to participate, you can withdraw your consent at any time without giving a reason. All information about your will then be anonymized. There will be no consequences for you if you choose to not participate or later decide to withdraw.

Your personal privacy – how we will store and use your personal data
We will only use your data for the purpose(s) specified in this information letter. We will process your personal data confidentially and in accordance with data protection legislation (the General Data Protection Regulation and Personal Data Act).

- The only persons who will have access to the personal data will be the master's student Lucas Paruch and Dr. Viktoria Stray at the University of Oslo
- All personal data will be treated confidentially. Personal data will be stored in a separate file and protected with password and login
- No names, addresses, e-mails, or other personal data will be used in the publication resulting from this project.

What will happen to your personal data at the end of the research project?
The project is scheduled to end May 2020. Once the project is finalized, all personal data will be deleted in such a way that the only available data will be anonymous. All digital recordings will be deleted.

Your rights
So long as you can be identified in the collected data, you have the right to:

- Access the personal data that is being processed about you
- Request that your personal data is deleted
- Request that incorrect personal data about you is corrected/rectified
- Receive a copy of your personal data (data portability), and
- Send a complaint to the Data Protection Officer or The Norwegian Data Protection Authority regarding the processing of your personal data

What gives us the right to process your personal data?
We will process your personal data based on your consent.

Based on an agreement with University of Oslo, Department of Informatics, NSD – The Norwegian Center for Research Data AS has assessed that the processing of personal data in this project is in accordance with data protection legislation

Where can I find out more?
If you have questions about the project, or want to exercise your rights, contact:

- *University of Oslo* via *associate Professor Viktoria Stray (stray@ifi.uio.no , +47 93610848)*
- Our Data Protection Officer: *Maren Magnus Voll (personvernombud@uio.no)*
- NSD – The Norwegian Center for Research Data AS (personverntjenester@nsd.no , +4755582117)

Yours sincerely,

Lucas Paruch
Master's student

Consent form

I have received and understood information about the project *Software testing in agile development* and have been given the opportunity to ask questions.

I give consent to participate in observations and interviews.

I give consent for my personal data to be processed until the end date of the project, approx. May 2020.

(Signed by participant, date)

B Interview Guide for Testers

Information before the interview

Thank the participant for taking his/her time to be interviewed

Present myself to the participant (name, background, goal of the interview)

Inform the participant about privacy policy and voluntary participation

- Data gathered from the interviews will be used to answer the research question
- Any personal data will be treated confidentially
- All participants will be anonymized
- It is voluntary to participate in the interview, and the participant can at any time withdraw his/her consent

Inform the participant about the usage of audio recording

- Ask if such a device could be used during the interview
- No one else other than the interviewer will have access to the recording, the audio will be deleted following the delivery of the thesis

Estimate the length of the interview (45-90 minutes)

Background

Can you tell me about your role, and what has generally been your tasks/assignments?

Can you shortly explain what the project is about?

How many years of experience do you have as a tester?

How long have you worked on this project?

How long have you worked at the consultancy company?

- Have you worked elsewhere before?
    - If yes, why did you switch?

How many team members are there in the team?

What kind of software development process do you use for the project?

Collaboration

To what degree do you feel 'connected/associated' to the team?

Who do you have the most contact with, within the team?

How do you feel about the team's coordination ability?

Problem-solving

Have there been challenges with working with others in the team?

- If yes, who was it with?
- What happened?
- Have there been any countermeasures after the incident?

Secondary knowledge

How much knowledge do you have about the other team members' fields?

- Is it easier to do your work when you know what the others are working with (in a specific sense)?
- Do you at times do work tasks meant for other team members?

Could you imagine to switch to something other than your current role?

- Within the field of testing? In another field?
    - Why/Why not?

Skills / learnability

Do you feel like there are skills that you currently lack in order to do your job properly?

- How would you proceed to learn a skill?
    - Have you tried to learn something new at your own discretion?
        - How did you start? What resources did you utilize?
- Who would you have spoken with if you wished to improve an existing skill?
- How would you proceed to help someone else learn a skill?

Communication

How much time do you spend on interaction with others to complete a task?

Assume that you have found a bug during testing, how would you present it to the team?

- What if someone says it is not a bug? How would you convince them?

Do you feel that the ability to communicate is more important than technical abilities?

Structure and effectiveness

In general, how structured are you (both during work and off-work)?

Do you think being structured is something that is needed to become a tester?

- Why/why not?

How would you define an effective tester?

Is there a correlation between effectivity and how structured one is?

Stress

Do you generally feel stressed during work?

Can you give me an example on a day where you were super stressed?

Do you operate better under stress/pressure?

Are you more stress resistance now, compared to when you first started working?

Adaptability/reflectiveness

Have there been cases where you had to reprioritize a task or do something else than what you had planned in your workday?

- Did this decision come from the product manager, the customer, or the team as a whole? Have you ever somewhat doubted their decision?

How much pre-work do you perform before starting at an assignment?

- Do you tend to think a lot before?

Attentiveness

Have there been cases where high severely bugs have slipped through the QA environment and onto the production?

- How did it happen?
- Were there any countermeasures taken after the incident?

Do you see things from another perspective than others?

- Have there been cases where you have pointed out things that no one has thought of? Are you usually like that?
- In what degree do you tend to remember minuscule details? (Use an example)

Creativity

How creative are you?

Do you get to use your creativity during work?

Do you think creativity is something needed in a tester?

- Why/why not?

Motivation

What keeps you going during work?

Have there been days where you perform better than other days?

- How/why?

Do you keep contact with other team members outside work-hours?

Do you often attempt to get to know new people (both during work and outside)?

Profession

Do you feel that software testing is looked down upon in regards to other roles in a software development project?

What attributes do you think a software tester should have?

Closing inquiries

Have you ever worked at a non-agile software tester?

- If so, how has the experience differed from your current ways of working?

How do you foresee the evolution of software testing in the future?

- Will software testing become more embedded within roles?
- Do we need dedicated testers?
- Evolution of the software testing role

If you could have started all over, would you still have chosen the career path that led you to where you currently are?

Is there anything we haven't discussed that you would like to add?

C Interview Guide for Non-testers

Information before the interview

Thank the participant for taking his/her time to be interviewed

Present myself to the participant (name, background, goal of the interview)

Inform the participant about privacy policy and voluntary participation

- Data gathered from the interviews will be used to answer the research question
- Any personal data will be treated confidentially
- All participants will be anonymized
- It is voluntary to participate in the interview, and the participant can at any time withdraw his/her consent

Inform the participant about the usage of audio recording

- Ask if such a device could be used during the interview
- No one else other than the interviewer will have access to the recording, the audio will be deleted following the delivery of the thesis

Estimate the length of the interview (20-45 minutes)

Can you tell me about your role, and what has generally been your tasks/assignments?

Please tell me briefly about the development project

How many years of experience do you have as X?

How many members are there in the team?

What is your relation to the software tester? (In what way do you interact with him/her?)

What knowledge do you have about the field of software testing?

- Do you sometimes do testing tasks?

Has the software tester helped you with your work tasks?

- How so?

To what degree do you have to look at the whole picture? (Domain knowledge, technical architectures etc.)

- How is that different from the software tester, according to your experience?

Are there lots of communication between you/team and the software tester?

- How?

How do you feel about the tester's communicative abilities? (Describing bugs, asking critical questions, etc.)

- Do you feel like they can improve on something?

How active are the testers in meetings?

How perceptive are the software testers?

- Are they attentive for details?
- Have they brought up things that no one else has thought of?

Do you see any correlation between creativity and software tester?

- How is it different from being a X?

Do you feel that a software tester needs to be more structured than a X?

- Why, why not?

Have there been days where you perform better without a tester?

- How about with a tester? Why?

Do you remember a situation where the software did a really good job?

- Can you describe it?
- What about the opposite?

How do you define an effective tester?

If you were to choose between a tester who isn't necessarily skilled in logic/technical but who is extremely good in communication and team coordination vs a tester who is vice versa, who would you prefer to work with?

- Why?

Which attribute do you think a tester should possess?

Do you feel that the software tester is connected to the team? (Cohesion)

What are your predictions for the evolution of software testing in the future?

- More embedded within teams?
- Do we still need dedicated testers?
- Evolution of the software testing role

Why do you think people want to be testers and not X?

Is there anything we haven't discussed that you would like to add?

D Observation Protocols

Location	• Workplace layout • Software testers' seating • Actors sitting in proximity of tester
Participant	• Name and role of employees • Behavior towards each other • Attitude towards tester • Tester's attitude towards others
Ceremonies	• Start-time, end-time, duration of meetings • Attendees and roles • Facilitator • Points of interest • Tester's behavior during meetings • Personal thoughts post-meetings
Informal interactions	• Who is talking? • Conversation topic • Classification of conversation (problem-solving, everyday life etc.)
Tester's attributes	• Emotions throughout the day • What does the tester do if unsure of doing a task? • Level of focus during work hours • Personality exhibit • Level of sociality

E EASE2020 Article

Characteristic Traits of Software Testers

Lucas Paruch
University of Oslo
Oslo, Norway
lucasp@ifi.uio.no

Viktoria Stray
University of Oslo
Oslo, Norway
stray@ifi.uio.no

Charlotte Bech Blindheim
Itera
Oslo, Norway
charlotte.bech.blindheim@itera.no

ABSTRACT

Although there has been extensive research on software testing technicalities - such as testing tools and practices - little research has been conducted within human factors of software testing. In collaboration with Itera, a consultancy company, we begin to fill this research gap. Our current qualitative data-set consists of observation notes, interview transcripts, and conversation logs. Our findings suggest that creativity, being structured, having the ability to see the whole picture, having good interpersonal skills, and eagerness-to-learn are desired traits for a software tester.

KEYWORDS

Human factors, traits, characteristics, software testing, testers, agile, software engineering

ACM Reference Format:
Lucas Paruch, Viktoria Stray, and Charlotte Bech Blindheim. 2020. Characteristic Traits of Software Testers. In *Evaluation and Assessment in Software Engineering (EASE 2020), April 15–17, 2020, Trondheim, Norway.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3383219.3383270

1 DESCRIPTION OF THE SETUP

The industrial experience described here is the current findings of collaboration with a consultancy company, Itera. Itera has over 500 employees across Europe and is involved in the banking and finance sector. There are currently nine consultants working as testers and test-leads in Norway. As such, current findings are based on the data collected from five of these consultants.

2 MOTIVATION FOR COLLABORATION

Within the software testing field, research focusing on human factors or the psychological aspect of testing is scarce [6, 8]. Moreover, the practitioner literature, such as the ISTQB foundation level syllabus - one of the most recognizable certifications within software testing - mentions little on the topic at hand [2]. We searched for papers focusing on human factors in the field of software testing in agile projects. A search in Scopus returned approximately 500 papers, where only 17 were relevant studies focusing on the topic in the period 2009 - 2019.

Itera's motivation for collaborating with researchers is to better understand how to support the testers they have employed, and what traits the company should look for when they hire new software testers. The company aims to get a better insight into how testing is conducted so that they can improve continuously. They want to give their testers autonomy to think differently, be more creative, and become more proficient in testing skills.

3 METHODS AND MODES OF ENGAGEMENT

So far, we have conducted five interviews with software testers in the company. These testers' working experience range from one year to six years. We have observed participants working, and we also have access to their communication on Slack. The data-sets, therefore, consists of observation notes, interviews, and chat logs. On the days of observation, the first author noted down timestamps and points of interest with focus on the tester within the project.

Moreover, interviews were held both within and external from the project of observation. These ranged approximately from 45 minutes to 1 hour and were held solely with testers from the consultancy company. All interviews have been transcribed. Logs from communication tools were also used in order to extract additional support for the data collected through observations and interviews, such as follow-ups or elaborations. We analysed the data using thematic analysis [3].

4 TRAITS OF THE SOFTWARE TESTER

Our analysis of the data material revealed five characteristic traits of the software testers, which we will describe in this section.

A trait that all participants mentioned was creativity. The interviewees mentioned the importance of being creative in order to find abnormal bugs. One participant stated *"I've managed to find weird bugs by being creative, such as mid-way force shutdowns and performing unusual process-sequences. One has to test like that because the users are always creative"*. Several participants also mentioned curiosity as an essential trait, such as being curious of domain knowledge and question the requirements' reasoning. One participant described that curiosity allowed him to learn domain knowledge swiftly, and mentioned that his knowledge even surpassed the product owner's.

Additionally, all five participants shared a common trait of being structured. One participant recalled *"Every Friday, I look at the calendar for what's happening next week. Once a month, I also look through the calendar and see what's going to happen next month, so that I may plan ahead"*. Another participant added *"I can't function if there is no structure. I need to have everything noted down in my calendar, and to-do's needs to be written down in checklists and notes.*

Moreover, the ability to understand the big picture was something that was discovered from every participant; they stated it was focal for testers to look at things as a whole and see the connections between them. One participant stated *"Often as a developer, you receive a task and you do it. They are not very involved in the*

EASE 2020, April 15–17, 2020, Trondheim, Norway

ACM ISBN 978-1-4503-7731-7/20/04…$15.00
https://doi.org/10.1145/3383219.3383270

entire process. I think it's more exciting to be involved in both the functional and technical aspects, not just super-technical". Another said *"One should be somewhat technical proficient, but that is not most important. The most important part is that you need to be able to make sense of which part of the system is prone to bugs".*

All participants mentioned the importance of being friendly and providing constructive feedback to other members, since testers are often the bearers of bad news. Constructive feedback somehow mitigates negative dynamics between testers and colleagues in the team. For example, one participant mentioned that *"I was on a project few years back where I sat next to the developers. When I found a bug, I stood up and walked towards them with a friendly smile".* Another explained that he gave explanations on what went wrong and did not focus on whose fault it was. A third mentioned that *"Whenever I find a bug, I go to the developer in mind and ask him if it's suppose to be like that. I try not to point any fingers because that's never pleasant for anyone and it's not appreciated".* We observed that testers were thorough with the way they reported bugs in meetings by the respective digital task board, one participant was particularly thorough with describing what went wrong, the expected result and actual result, and the steps he took to reproduce it.

Lastly, participants mentioned that their motivation during work was triggered by continually learning something new, having fun with pleasant co-workers, and performing enjoyable yet challenging tasks. One participant, when asked what he liked about his job, stated *"Lots of things motivate me during work; cool projects, challenging workdays - those that one actually has to use the head - and also working with lots of different people, which is always fun"* Another mentioned *"It's fun to work in a team, and work together with others. When I have issues, there's a high probability someone else on the team has the solution. Sometimes I'm the one that helps them. I think it's really cool to work this way, and it becomes a lot more social during work - I look forward to work every day".*

5 LESSONS LEARNED AND IMPLICATIONS

Based on the current results, being creative and structured are the most important traits for a tester to possess. Burnstein [4] also suggests that testers need to be creative and experiment-oriented, while Kanij et al. [11] found that software testers tended to be more organized, disciplined, and hard-working.

Communication skills are especially important for software testers [6], and reporting faults in a constructive way is advisable in order to foster a synergized work environment. Ahmed et al. [1] described software testers as "the software development team's worst enemy" and therefore they need good interpersonal skills. A recent study shows that people are more careful in their communication if a conflict is thought to occur [9]. Software testers need to communicate in a way that does not provoke conflict within the team. It was noted that two of the testers became friends with the rest of the team, to the point where the team began to socialize during off-work hours. They stated that it became easier to ask for help and seek feedback after the get-togethers. Psychological safety has been found to improve team performance in agile projects [10], and making software testers less worried about offending developers when confronting them with bugs [14]. Current findings add to the understanding that psychological safety is important and may occur through social interactions among team members.

We found curiosity and the ability to see totality as important traits for testers. A recent study on the skills of software testers also found that testers need to have both broad views and attention to detail [7]. Two existing studies focused on motivational factors for testers [5, 13], found that enjoying challenges and having work variety were included in the motivational factors - our current findings suggest confirming this.

Some of the traits we discovered as essential for software testers, such as the ability to engage with others, having decision-making skills, and eagerness-to-learn has also been identified as important for software developers [12]. Future work could further investigate differences and similarities in traits, as well as how the degree of importance varies, for people having these two types of roles.

ACKNOWLEDGMENTS

We would like to thank all the participants that agreed to be observed and interviewed. A special thanks goes to Itera for making this collaboration setup possible.

REFERENCES

[1] F. Ahmed, L. F. Capretz, and P. Campbell. 2012. Evaluating the Demand for Soft Skills in Software Development. *IT Professional* 14, 1 (Jan 2012), 44–49. https://doi.org/10.1109/MITP.2012.7

[2] Rex Black, Erik van Veenendaal, and Dorothy Graham. 2020. *Foundations of Software Testing ISTQB Certification* (4th ed.). Cengage Learning EMEA; 4th edition.

[3] Virginia Braun, Victoria Clarke, Nikki Hayfield, and Gareth Terry. [n.d.]. Thematic Analysis. In *Handbook of Research Methods in Health Social Sciences*, Pranee Liamputtong (Ed.). Springer, 843–860. https://doi.org/10.1007/978-981-10-5251-4_103

[4] Ilene Burnstein. 2006. *Practical software testing: a process-oriented approach.* Springer Science & Business Media.

[5] Anca Deak, Tor Stålhane, and Guttorm Sindre. 2016. Challenges and strategies for motivating software testing personnel. 73 (2016), 1–15. https://doi.org/10.1016/j.infsof.2016.01.002

[6] Raluca Florea and Viktoria Stray. 2018. Software Tester, We Want to Hire You! an Analysis of the Demand for Soft Skills. In *Agile Processes in Software Engineering and Extreme Programming (LNBIP)*, Juan Garbajosa, Xiaofeng Wang, and Ademar Aguiar (Eds.). Springer International Publishing, 54–67.

[7] Raluca Florea and Viktoria Stray. 2019. The skills that employers look for in software testers. *Software Quality Journal* 27, 4 (2019), 1449–1479.

[8] Vahid Garousi and Mika V. Mäntylä. 2016. A systematic literature review of literature reviews in software testing. 80 (2016), 195–216. http://www.sciencedirect.com/science/article/pii/S0950584916301446

[9] Lucas Gren. 2017. The Links Between Agile Practices, Interpersonal Conflict, and Perceived Productivity. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17)*. ACM, 292–297. https://doi.org/10.1145/3084226.3084269 event-place: Karlskrona, Sweden.

[10] Tomas Gustavsson. 2018. Impacts on team performance in large-scale agile software development, Vol. 2218. CEUR-WS, 421–431. http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-70241

[11] T. Kanij, R. Merkel, and J. Grundy. 2015. An Empirical Investigation of Personality Traits of Software Testers. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. 1–7. https://doi.org/10.1109/CHASE.2015.7

[12] Paul Luo Li, Amy J Ko, and Andrew Begel. 2019. What distinguishes great software engineers? *Empirical Software Engineering* (2019), 1–31.

[13] R. E. d S. Santos, C. V. C. d Magalhães, J. d S. Correia-Neto, F. Q. B. d Silva, L. F. Capretz, and R. Souza. 2017. Would You Like to Motivate Software Testers? Ask Them How. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 95–104.

[14] Viktoria Stray, Tor Erlend Fægri, and Nils Brede Moe. 2016. Exploring norms in agile software teams. In *International Conference on Product-Focused Software Process Improvement*. Springer, 458–467.