OULUN AMMATTIKORKEAKOULU

Toni Kemppainen

The benefits of Test Automation in Software Development

The benefits of Test Automation in Software Development

Toni Kemppainen
Thesis
Spring 2022
Degree Programme in Information Technology
Oulu University of Applied Sciences

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, Ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä: Toni Kemppainen
Opinnäytetyön nimi: The benefits of Test Automation in Software Development
Työn ohjaaja: Janne Haka-Risku
Työn Tarkastaja: Eino Niemi
Työn valmistumislukukausi ja -vuosi: Kevät 2022
Sivumäärä: 41

Opinnäytetyö toteutettiin yritykselle nimeltä Combitech, joka on pohjoismainen tekniseen konsultointiin erikoistunut yritys. Combitech on osa kansainvälistä Saab-konsernia ja sillä on yli 2100 työntekijää Suomessa, Ruotsissa, Norjassa ja Tanskassa. Suomessa yritys on erikoistunut kyberturvallisuuden ja puolustuksen kehittyneisiin teknisiin ratkaisuihin ja palveluihin.

Dokumentti on muotoiltu noudattaen APA-tyyliin kuuluvaa ohjeistusta ja sen rakenne mukailee Runesonin ja Höstin (2008) tutkielman sivulla 157, Kitchenhamin ym. (2008) esittelemää tapaustutkimusten raporttipohjaa. Johdanto on muotoiltu CARS-mallin mukaan ensin määrittelemällä tutkimusaihe, esittämällä motivaatio tutkimustarpeelle ja sijoittamalla opinnäytetyön tutkimus tähän kenttään. Kaikkien lähteeksi merkittyjen asiakirjojen julkaisukanavan taso on tarkistettu Tieteellisten seurain valtuuskunnan ylläpitämän Julkaisufoorumin hakukonetta käyttäen ja suurin osa niistä on myös vertaisarvioitu.

Opinnäytetyö alkaa johdannolla, johon kuuluu työn taustan esittely, työn päämäärä, ratkaistavan ongelman kuvaus ja tutkimustavoitteet. Toisessa kappaleessa esitellään aikaisempia tutkimustöitä aiheesta, avataan erilaisia käsitteitä, esitellään testaukseen liittyviä käytänteitä ja työkaluja, sekä pohditaan testaukseen liittyvien kustannusten arviointia.

Opinnäytetyön kolmas kappale esittelee tapaustutkimuksen tutkimuskysymykset, esittelee tutkimuksen kohteena olevan järjestelmän ja esittelee datan keräyksen menetelmät. Tutkimuksen kohteeksi valittiin eräs toimeksiantajayrityksen järjestelmä. Datan keräys tapahtui haastatteluiden muodossa sekä järjestelmän kehitystyössä käytettävistä työkaluista kuten Jira, Jenkins ja SonarQube. Haastattelut toteutettiin kaksivaiheisesti, ensin haastattelemalla paikallisen yliopiston työntekijää ja vastausten perusteella muotoilluilla kysymyksillä toimeksiantajayrityksen työntekijää. Määrällistä tietoa kerättiin pääasiassa lähdekoodin testikattavuudesta, asiakkaiden raportoimista bugeista ja manuaalisten testien suoritusajoista.

Neljäs kappale sisältää edellisen kappaleen suunnitelmien mukaan tuotettujen tutkimusten tuloksia. Haastatteluista kävi ilmi, että akatemian edustaja ja yrityksen edustaja molemmat ajattelivat testiautomaation olevan erillinen manuaalisesta testauksesta johtuen eroista suoritustiheydessä. Testikattavuudesta ja asiakkaan raportoimista bugeista tuotettujen kaavioiden perusteella voitiin huomata customer issueiden määrän laskevan testikattavuuden noustessa. Manuaalisten testien suoritusajat olivat keskimäärin 26,5 minuuttia, saman

testitapauksen automatisoinnin viemä keskimääräinen aika 408,9 minuuttia ja break even -piste 19,4 suorituskertaa. Tuloksista voitiin päätellä, että suurin koituva hyöty automatisoinnista oli huomattavasti suurempi suoritustiheys.

Viides kappale kokoaa yhteen tutkimuksen tuloksista saadut johtopäätökset ja antaa ehdotelmia tuleville, tämän opinnäytetyön pohjalle tehtäville tutkimuksille.

Asiasanat: Ohjelmistotestaus, Testiautomaatio, laadunvarmistus, IAC, Sijoitetun pääoman tuottoaste, kriittisen pisteen analyysi, Tuotantomahdollisuuksien käyrä.

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author: Toni Kemppainen
Title of thesis: The benefits of Test Automation in Software Development
Supervisor: Janne Haka-Risku
Examiner: Eino Niemi
Term and year when the thesis was submitted: Spring 2022
The number of pages: 41

The thesis was written in collaboration with Combitech, a Nordic company specialising in technical consulting. Combitech is part of the international Saab Group and has more than 2,100 employees in Finland, Sweden, Norway, and Denmark. The company specialises in advanced technology solutions and Finland's cyber security and defence services.

The document is formatted according to APA-style guidelines and follows the case study report template structure presented by Runeson and Höst (2008) on page 157 by Kichenham et al. (2008). The introduction is formulated according to the CARS model by first defining the research topic, presenting the motive requirement for the research need and placing the research of the thesis in this field. The level of the publication channel of all documents marked as a source has been checked using the search ePublication forum's search engine by the Delegation of Scientific Societies. Most of them have also been peer-reviewed.

The third part of the thesis presents the research questions of the case study, introduces the system under study and presents the methods of data collection. A system of the client company was selected for the study. Data were collected in the form of interviews and from tools used in the development of the system, such as Jira, Jenkins and SonarQube. The interviews were conducted in two stages, first by interviewing an employee of the local university and by asking questions to the employee of the client company. Quantitative data was collected primarily on source code test coverage, bugs reported by customers, and manual test execution times.

The fourth section contains the previous section's results of the studies produced as planned. The interviews revealed that a representative of the Academy and a representative of the company both thought that test automation was separate from manual testing due to differences in performance. Based on the charts generated from test coverage and bugs reported by the customer, it could be seen that the number of customer issues decreased as test coverage increased. The average execution times for manual tests were 26.5 minutes, the average time taken to automate the same test case was 408.9 minutes, and the break-even point was 19.4 times. From the results, it could be concluded that the main benefit of automation was a significantly higher performance density.

The fifth chapter summarises the study's conclusions and provides suggestions for future research based on this thesis.

Keywords: Software testing, Test automation, Quality Assurance, IAC, Return on Investment, Break-even analysis, Production Possibilities Frontier.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

BBT	Black Box Testing
BEP	Break-even Point
CBA	Cost-benefit analysis
CBA	Cost-Benefit Analysis
COI	Cost of Investment
IAC	Industry-Academia Collaboration
ICSE	International Conferences on Software Engineering
JQL	Jira Query Language
MBT	Model-Based Testing
PPF	Production Possibilities Frontier
QA	Quality assurance
ROI	Return on Investment
SE	Software Engineering
SUT	System Under Test
TA	Test Automation
TDD	Test-driven development
WBT	White Box Testing

1 INTRODUCTION

1.1 Background

Software testing has been an active research area for many decades, and today quality engineers can benefit from many results, tools, and techniques (Luciano & Mauro, 2006, p. 110). The industry's implementation of software testing tools and testing methodologies has been on a steady climb for years. The market value of software testing is growing each year and so is the time and resources companies have to spend on testing. Ever increasing price of software testing raises a question about how tangible are the benefits testing creates for the company. In addition, there is also a disjoint between industry and academic focus areas in software testing. Garousi and Felderer (2007, p. 7) reported that analysis of talk titles in industrial versus research conferences showed that practitioners and researchers are focusing on quite different things in testing. Having different focus areas is a major reason which leads to (very) low relative IAC in software testing, and also other areas of software engineering, in almost every country.

Many case studies have been made on the effects of software testing by students and researchers, but many of these studies were published several years ago, and the testing methods have changed drastically. Hence, room exists for case studies conducted in the 2020s to provide more recent information.

1.2 Goal and structure of the thesis

The case study reported in this thesis follows the "Guidelines for conducting and reporting case study research in software engineering" by Runeson and Höst (2008). The report is structured, as proposed by Yin (2003, p. 156), Linear-analytic the standard research report structure (problem, related work, methods, analysis, conclusions).

This thesis aims to study the benefits of manual and automated testing in software development. The thesis is divided into two parts: A theoretical part comprising a literature review of the previous research on the benefits of TA in software development and an in-depth empirical case study of enhancing the target system's TA. The review focuses on the academic literature and other empirical studies. The case study seeks to

clarify the value TA can provide to the company from a business perspective using methodologies presented in the academic research. The essential methods and formulas are presented in chapter 1.3. Qualitative and quantitative data provide a deeper understanding of the topic. The qualitative data is gathered in a personal interview. The quantitative information is mainly collected from the software and the various tools used in the development process.

1.3 Problem statement

Software testing is one of the most significant expenses related to software development. Highest estimates of the relative cost of testing range up to 60% of the total costs of the development process. This puts pressure on the companies to assess more cost structures and re-evaluate how much resources to allocate to testing. Another issue facing the field is low industry-academia collaboration. Researchers and industry practitioners disagree on what to focus on in software testing. Both problems give rise to the necessity for further empirical study, which this thesis aims to fulfil.

1.4 Research objectives

The objective is to review some existing research studying the benefits of TA, compile the results and add to their findings by collecting and analysing data while working for a medium-sized tech company. The studied case is the testing infrastructure and the development work of SUT, which is part of a much larger software project developed by the same company. Documented and proven techniques derived from previous research on the topic, especially those focusing on TA's objective pros and cons, form the theoretical basis for this study. The case study aims to find monetary benefits for test automation and software testing in general. The research questions of this study are defined in chapter 1.1.

2 RELATED WORK

This chapter introduces literature on critical concepts brought up in the study, the cost of software testing, differences between manual and automated testing, previews research on the benefits of TA and the current state of the research in software testing.

2.1 Software testing

Software testing consists of dynamically verifying a program's behavior on a finite set of test cases—suitably selected from the usually infinite domain of executions—against the specified expected behavior (Bourque & Fairley, 2014, p. 42). Based on the design technique, tests can be divided into two broad categories: black box and white box. For a given particular system under test (SUT), many combinations of tests are possible (e.g., manual functional black box tests, non-functional automated system tests, etc.). The decision of which tests to use is usually taken by the software tester and documented in a test plan based on predefined test objectives and quality factors (e.g., correctness, reliability, efficiency, usability) as well as budget, time, and resources available. Commonly, a combination of several test types is used. In recent years, there has been a push for automation, as new techniques to automate test design and execution are proposed, and commercial tools that support automated testing proliferate. (Dobles et al. 2019, p. 1.) Heusser and Kulkarni (2012) list five traditional phases included in the software testing process: 1. Test planning, 2. Test case design and construction, 3. Test data creation, 4. Test execution, 5. Test closure.

2.1.1 Test automation (TA)

Automation is the integration of testing tools into the test environment in such a manner that the test execution, logging, and comparison of results are done with little human intervention (Koirala & Sheikh, 2009, p. 127). Compared to manual testing, TA requires process-oriented work. Manual testing only requires for the tester to execute the given test case, whereas automated tests have to be designed, developed, and maintained as a process. Sneha and Malle (2017, p. 77) recognize four activities included in the automation testing

process: 1. Test plan, 2. Test design or Implementation, 3. Execution of the tests, 4. Test evaluation and analysis.

2.1.2 Testing tools

Automated testing tools assist software engineers to gauge the quality of software by automating the mechanical aspects of the software-testing task. Automated testing tools vary in their underlying approach, quality, and ease-of-use, among other characteristics. In addition, the selection of testing tools needs to be predicated on characteristics of the software component to be tested (Michael et al., 2002, p. 1.)

Strategic Planning and Economic Analysis Group (2002, s. 1, p. 11.) reported that there is a lack of readily available performance metrics, procedures, and tools to support software testing. If these infratechnologies were available, the costs of performance certification programs would decline and the quality of software would increase. This would lead to not only better testing for existing products, but also to the testing of products that are not currently tested. Imtiaz et al. (2019, p. 211) in their study found several papers on TA and automation tools, but no paper focused in detail on the test automation tool adaption. This area becomes increasingly important as the need for software test automation increases due to pressure to perform rapid or continuous releases.

2.1.3 Cost of testing

Software testing is a rigorous and costly process that, in many cases, requires management's approval. Because the benefits of software testing are less immediate and apparent than actual software development, the executives may be hesitant to divert teams' limited resources away from what generates the most value from the customer's perspective. Alégroth et al. (2016, p. 1) point out in their study that both academia and industry report testing to be sometimes upwards of 50 per cent of total development cost and rarely below 20 per cent. Kumar and Mishra (2016, p. 14) estimate this to be even higher, with different types of testing being nearly 60% of the total software cost.

Despite the tremendous investment in testing, its benefits can often go unnoticed. Regardless of the fact, doing no testing is rarely a good idea. When companies invest little or no in testing, the adverse effects become observable across the software development process. In the worst-case scenario, this can lead to customers finding more errors in the production version of the software. Errors detected this late cost multiple times more to fix than in the early stage of development, and sometimes they can cause irreparable damage to the client's environment and data. Inadequate testing has already led to increased expenditure caused by software errors worldwide. Garousi and Mäntylä (2016, p. 92) reported that a 2013 study by Cambridge University states that the global cost of locating and removing bugs from software has risen to $312 billion annually. It makes up half of the development time of the average project.

When calculating the cost of automated and manual testing, it is feasible to consider the savings from catching the software bugs early in the development cycle. In other words: the cost of not covering error-prone code with tests. The methods included in chapter 3.3 aim to calculate the net benefits of testing with that in mind. However, this cost analysis is more effortless for manual testing, where the costs increase linearly. Test automation costs should be studied separately from manual testing because they consume resources differently. To illustrate the cumulative cost of manual testing, only two variables are needed: the hourly cost of labour and the average time it takes to execute a test case. Charges related to TA are harder to illustrate and require too many variables in a broader time frame to be generalised in a single graph. For this reason, Figure 1 only compares the implementation cost of automating the same test case instead of executing it manually. The reference value for the average time it takes to complete one test case manually is taken from historical data, as shown in Table 1, compiled in the case study by Aranha and Borba (2009, p. 170).

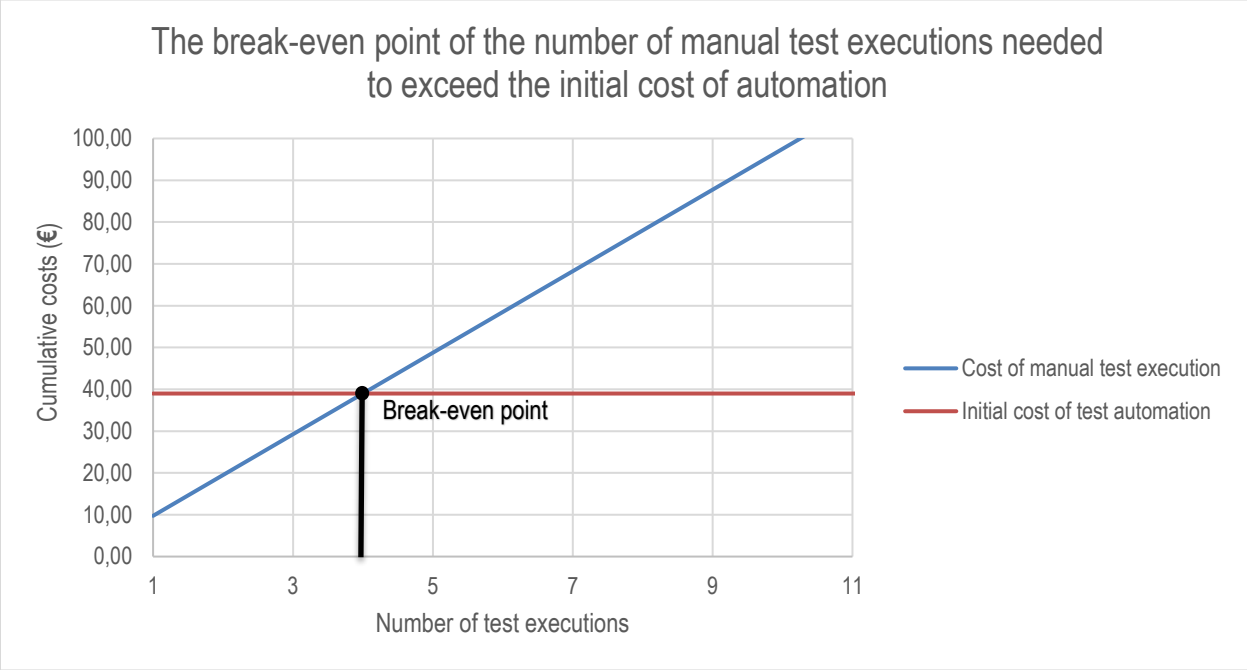
TABLE 1. (Aranha & Borba 2009) Descriptive Statistics of the Historical Database Used in the Study

Test Suite Characteristic	Mean	Median	Min	Max	Std Dev
Number of tests	48.55	57	15	104	21.37
Execution points	14,935.12	15,553	6,086	28,174	4,641.48
Effort (hh: mm)	07:05	06:56	02:28	14:40	02:35

The average execution time of 8.75 minutes, calculated by dividing the historical data's Mean Effort (hh: mm) by the Mean Number of tests in a test, suite implies the test cases are primarily simple and comparable to

unit tests. For the average hourly testing cost, the study references the in-house price of one developer hour of 65 euros calculated in the case study by Smite and Solingen (2016, p. 64). Results for 10 test execution using the two variables is shown in Figure 1. Execution time was rounded to 9 minutes for the sake of convenience. For the implementation cost of automating the test case, the graph uses the same fictitious multiplier of 4 as in Figure 2 of the conference paper by Ramler and Wolfmaier (2006, p. 87).

FIGURE 1. Illustration of the cumulative costs generated by both manual execution and the initial test automation investment



The resulting figure is essential since it can quickly illustrate how the costs related to both manual and automated testing are expected to behave during the design phase in chapter 3 and the analysis phase in chapter 4 with actual data. All three variables, hourly testing cost, manual test execution time and implementation cost of TA, vary greatly depending on the type of software and testing practices that it's not possible to draw a general conclusions applying all test suites. Figure 1 also assumes no maintenance costs are included in the automated test case.

Figure 1 illustrates how the first automated test runs require more resources than testers manually running them. Once the test script is stable and can be reliably used to replace the human tester, execution costs drop to a fraction of testing by hand. The break-even point is the number of test executions, where the overall

cost of manual testing (T_m) exceeds the cost of automating (T_a) the same test case. What Figure 1 does not display is the long-run maintenance costs of TA. Berner et al. (2005, p. 577) stated that testware has to be maintained with each new release of the system under test. Depending on the frequency of release cycles and the lifespan of the system under test, its maintenance tends to have a much bigger impact on the overall cost for testing than the initial implementation of automated tests. Testware architecture therefore has to be designed to minimize maintenance cost.

Sometimes automation takes longer to break even, and testers might want to extend their expectation to multiple test cycles in more than one project. A study by Jayachandran (2005, p. 36) found out that the average savings automated test can bring over manual testing per test cycle was $12.375. Jayachandran states that with a low profit margin and a high fixed cost . . . it does not make sense to hope for a positive return on investment from the testing cycles of the current project alone. Rather, the best option would be to amortize the fixed cost over several testing projects. The average profit margin of several testing cycles (spread across several testing projects) can be calculated by summing up their individual profit margins and dividing it by the total number of testing cycles minus one. The empirical study in chapter 3 of this thesis will also present a way to calculate the time and money saved per test cycle for automating test cases using the data collected from SUT.

The most direct way to decrease testing-related costs is to save developers' and testers' time. When measuring the hours kept, it is best to consider the whole software development process broadly. Solingen and Berghout (1999, p. 13.) stated that decreasing the costs of software development is primarily realised by decreasing the software development labour, and therefore cost reduction will mostly aim at executing software development processes more efficiently. A first step to this is identifying current effort and expenditure. Examples from practice in which costs are expressed are costs per source line, costs per function point, costs per life-cycle phase, or costs per product (subsystem). Software measurement is an excellent tool to start improving on this type of data. The focus will not only be on 'doing things better' but also on 'doing the right-things', because eliminating unnecessary activities will be one of the most interesting cost cutting activities.

Another way to reduce testing costs is to change testing and development practices. Recent industrial experience has shown that the use of MBT can lead to significant reductions in the cost of testing (Grieskamp et

al. 2011, p. 1). Strategic Planning and Economic Analysis Group (2002, s. 1, p. 12) reported that early detection of defects can greatly reduce costs.

2.1.4 To automate or to manually execute?

Manual and automated tests differ significantly in terms of requirements. The total cost produced by manually executed tests grows linearly to execution times, and the cost per execution is also an order of magnitude higher than for TA. However, their required initial cost is non-existent compared to automated tests. As reported by Dobles et al. (2019, p. 5): automated testing needs a higher initial effort, mainly caused by the creation of the scripts, but this cost can be amortized in time as automated tests are executed multiple times for regression testing.

Even though automating test cases can create significant savings, not all tests should be automated, and some test cases are not possible to automate within reasonable limits. The lifecycle of automated test cases is also limited due to the changes in the software, whereas manual tests can be comfortably executed as long as the tested functionality exists. In large codebases, the upkeep cost of TA can rival the costs created by the actual development work. Testers spend more time maintaining existing tests than creating new ones as the software grows. Kaner et al. (2001, p. 90) stated that: Testers invest in regression automation with the expectation that the investment will pay off in due time. Sadly, they often find that the tests stop working sooner than expected.

- The tests are out of sync with the product.
- They demand repair.
- They are no longer helping find bugs.
- Testers reasonably respond by updating the tests.

But this can be harder than expected. Many automators have found themselves spending too much time diagnosing test failures and repairing decayed tests.

However, all regression tests executed multiple times are generally good to automate, even though automated tests have a limited lifespan. Testers can abandon some of the broken tests, for example, when the

underlying code changes so much that it is impossible to use the same test script. It still saves time and resources needed to test the given functionality. TA can also bring some extra benefits compared to manual testing, such as more frequent test runs, all test runs being identical to one another and finding bugs that go unnoticed by a human tester. In addition, some functionalities are unrealistic to test by hand or require too many resources. Ramler and Wolfmaier (2006, p. 90) give an example of such test case in their conference proceeding: Consider for example performance and stress testing with 100 concurrent transactions accessing the same data. These types of testing require automation as it is impossible (or only at an unreasonable expense) to simulate such a scenario by hand. In contrast, manual testing is preferable to assess "intangible" requirements such as the aesthetic appearance and attractiveness of a user interface or Web site.

2.2 Quality Assurance

A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements (ANSI/IEEE Std. 730 —1981, p. 41).

2.3 Test Oracle

Barr et al. (2015, p. 2) define Test Oracle as a procedure that distinguishes between the correct and incorrect behaviors of the System Under Test (SUT). Test oracle can be a documented algorithm that allows human to calculate the correct output for given input, a separate program or a part of a software that always produces the same output for the same input, documentation that gives correct output for the given input, an expert who can tell by the output whether it is correct, or any other way to tell the output is correct. Although unlimited ways to create a test oracle exist, it is not possible to have an oracle for every test case. As claimed by Zhou et al. (2021, p. 1164): In many situations, an oracle is unavailable, or is theoretically available, but practically too expensive to be applied. This is known as the oracle problem, a fundamental challenge in software testing.

Ideally, a test oracle should have three capabilities that it can perform flawlessly: Generator, comparator, and evaluator. The generator provides expected results for each test. The comparator makes a comparison between obtained and predicted results. The evaluator determines whether the obtained and predicted results are close enough to pass.

2.4 Test-driven development (TDD)

Test-driven development (TDD), sometimes also called test-first (TF) programming, is a programming technique adopted in extreme programming (XP). The main idea behind TDD concept is that the programmer writes unit tests before writing any new code. When writing the test code, programmer in fact specifies how exactly the program would need to work in order to pass the test. The test necessarily fails, since there is no code yet. After that, the programmer writes the code needed to pass the test. When the test passes (and other previous tests pass too), the programmer looks at the code and if needed, improves its design. This is called refactoring. After that, a cycle is closed and the programmer proceeds with the next small piece of functionality. (Pančur & Ciglarič, 2011, p. 557).

2.5 Current state of the research

The following chapters discuss research conducted on software engineering and testing. Both chapters intend to determine how much study on the topics is done and how mature they are like fields of study. Important things to note are the number of scientific publications, how recent or relevant they are, and their impact in practice.

2.5.1 Software engineering

Software Engineering (SE) is an established discipline that has existed on its own for more than 40 years. The term software engineering first appeared in the 1968 NATO Software Engineering Conference and was meant to provoke thought regarding the perceived “software crisis” at the time. With the increasingly ubiquitous application of computing technologies, the maturity of SE has a considerable impact on almost every other discipline (i.e., it is hardly possible to imagine a modern society without heavy usage of software systems). Over the 40 years in consideration (1969-2009), the discipline of SE has gained a very substantial growth in depth and breadth of its research contributions. (Garousi & Ruhe, 2013, p. 2.)

2.5.2 Software testing

Today, software testing is a mature research field with thousands of journals, articles, conference papers, books, etc., released every year. Using the search term "software testing" in IEEEXplore, ACM digital library, Science Direct and Springer Link returned 991,860 publications for the years 2020-2022. As illustrated in Table 2, most of these publications were in the form of a research articles.

TABLE 2 Number of different types of publications on the topic "software testing" in the years 2020-2022

Research articles	811676	Review Article	577	Section	125
Chapter	77933	Editorials	563	Column	112
Conference Paper	55876	Front Matter	498	Reference Work	99
Review articles	16284	Discussion	416	Conference info	86
Short communications	14871	Software publications	409	Practice guidelines	86
Book	9259	Early Access Articles	403	Monograph	85
Book chapters	8531	Mini reviews	393	Introduction	74
Abstract	4958	Demonstration	377	Panel	67
Short Paper	4179	Editorial	332	Course	64
Protocol	3209	Survey	254	Errata	55
Reference Work Entry	2930	Opinion	232	Standards	39
Journals	1771	Work in Progress	221	Video Segment/Article	40
Conference Proceedings	1665	Tutorial	210	Book reviews	24
Poster	1384	Invited Talk	162	Examinations	15
Data articles	1089	News	160	Courses	2
Correspondence	1074	Magazines	159	Product reviews	1
Encyclopedia	939	Keynote	134	Replication studies	1
Case reports	882	Rapid Communication	134	Other	3304

Note. This table represents the number of different publications found when using the search engines: IEEEXplore, ACM digital library, Science Direct and Springer Link. The search term "software testing" was applied to the title, text and all metadata included in the publication.

Software testing as a field is actively studied by researchers, with over 400 thousand articles published each year as proof. However, only a tiny fraction of these is done in collaboration with the industry. IAC remains one of the significant steppingstones in the field. Garousi et al. (2016, p. 2) describe the situation: The global

software industry and the software engineering (SE) academia are two large communities. However, unfortunately, the level of joint industry-academia collaborations in SE is still relatively very low, compared to the amount of activity in each of the two communities. Garousi & Felderer (2017, p. 2) state that this is especially the case for software testing, which is a hot topic in SE research, reflected by the large amount of testing-related papers at recent ICSE conferences, and practice, reflected by the large amount and popularity of industrial testing conferences. The lack of mutual perception between industry and academia hurts both sides: researchers have fewer insights on the problems that are important to practitioners, while practitioners fail to learn what researchers have already discovered that might be useful to them . . . many SE practitioners and researchers and especially those in software testing are not talking to each other (adequately).

Researchers and practitioners have discussed various reasons for the lack of motivation for collaborations between industry and academia. Each side has different objectives, industrial problems lacking scientific novelty or challenges, and the solutions' low applicability and scalability problems developed in academia. (Garousi & Felderer, 2017, p. 2.) To bridge the gap between industry and academia and to foster IAC, a number of researchers from academia and also practitioners from industry have systematically studied and reported challenges, best practices (patterns for successful collaborations) and anti-patterns. (Garousi et al., 2016, p. 107.) Garousi et al. present five categories regarding IAC in SE with challenges and best practices documented for each: Problem formulation, Planning, Operationalization, Transfer and dissemination and Complete life cycle. For convenience reasons, this section will not go through all the listed challenges and best practices but lists the critical findings for every category below:

Problem formulation

i. The most common challenges during problem formulation are the differences between industry and academia in terms of time horizons, objectives, reward systems and perceptions of what is useful.
ii. The frequently-mentioned suggestions for addressing the mismatch and to get commitment are proper presentation and communication, and topic selection.
iii. Invest in order to understand the problems of industry to gain their commitment using systematic research approaches (should be done by both sides).

Planning

i. The difference in time horizon is the most generic challenge observed (more than one quarter of all included studies refer to the challenge).
ii. Other common challenges are limitations in planning to achieve high validity of the results and the challenge to achieve clear and realistic ambitions and goals in projects.
iii. To address i) and ii) common objectives and a common understanding have to be achieved, e.g. related to an awareness of different time horizons
iv. The most common research method suggested to consider in planning was case study.

Operationalization

i. The most frequently mentioned challenge is the lack of resources available from industry and academia side (25% of all included studies).
ii. Validity concerns need to be addressed when industry is involved.
iii. Four recommendations for the operational phase stand out. Approximately 50% of all studies recommend to base solutions based on real-world problems observed in industry. The key benefits need to be shown and demonstrated to the industry partner. The researcher should also be colocated at the company and have regular presence.
iv. Also, specific quality attributes of a solution have been identified that should be fulfilled, most commonly customizability.

Transfer and dissemination

i. The main hindrances in transfer are when research results are not demonstrated on an industrial scale, and there is a lack of availability of time to conduct the transfer.
ii. Practices important during operationalization are also important for technology transfer, in particular the ability to demonstrate benefits and achieving the important quality attributes mentioned earlier.

Complete life-cycle

i. The main challenge from the overall lifecycle perspective is the lack of relevance of research for practice.
ii. The most frequently mentioned practices (at least 25% of all included studies) to be successful in the overall end to end process were to run regular workshops and seminars, ensure management engagement, the need for champions, and to conduct the IAC in an agile way.

For the SE research community to have a meaningful future, there is a critical need to both industry and academia to collaborate with one another (Garousi et al., 2016).

3 CASE STUDY DESIGN

3.1 Research questions

This section covers the research questions for the case study. Questions are divided into one main question and sub-questions, which narrow down the subject for

- Research Question: What are the benefits of TA in software development?
- Sub-question 1: What are the critical issues in research?
- Sub-question 2: How does TA compare to manual testing?
- Sub-question 3: What monetary value can TA bring from a business perspective?
- Sub-question 4: How does the amount of testing correlate to the software quality?

3.2 Case and subject selection

A system consisting of multiple applications and testing infrastructure was selected for the case study's subject. Challenges related to the system currently include relatively low test coverage and noticeable time consumption of maintenance work such as fixing incorrect code. During the thesis, a lot of effort is put into developing the testing infrastructure while collecting information.

3.3 Data collection procedures

Triangulation is used in data collection procedures to ensure the information produced is versatile. To achieve this, both qualitative and quantitative methods collect data from the same phenomena. The qualitative research is done in the form of interviews. All technical data is collected using quantitative methods. Mainly third-order techniques, such as analysis of work databases and tool use logs, etc., are used. Git, Jira, Jenkins and SonarQube were selected as primary sources for quantitative data in the study's planning phase. The four selected tools include detailed information from several years back that can be easily exported to CSV form. For the data collection to be successful, each data point should be pinpointed to a specific moment in

time with accurate timestamps. The format used for timestamps is “dd.mm.yyyy HH:MM”, where dd = date, mm = month, yyyy = year, HH = hours and MM = minutes. These methods aim to find a correlation between changes in variables such as code coverage changes and the number of bugs found in production.

3.3.1 Interviews

The study includes interviews with both industry practitioners and members of academia. Interviews are carried out in two phases. The first one is a less structured interview with at least one member of academia. This is the essential part of the interviewing process since it can provide new information and improvement suggestions to the quantitative part of the study. The second interviewing phase is structured based on the answers received during the first interview and is done to industry practitioners. This way, the study attempts to bring relative insights from academia to the minds of professional developers and testers.

3.3.2 Filtering Jira xTest executions

By taking advantage of Jira’s query language (JQL), this method collects information on the test execution times of manually performed tests documented as Xray tickets. For this purpose, 13 tickets were chosen. Knowledge of how long it took to execute each test case was taken from xTest Executions’ “Started” and “Finished” -timestamps. This method is not always accurate since it only considers the time it took to mark every test step included in the Xray ticket as “PASS” and does not show how active the tester was at every given moment of the process. However, calculating the mean execution time using all test runs for each Xray ticket can estimate the actual time it takes to execute the given test case manually. This information becomes handy when analysing the break-even point for automation in chapter 3.4.2.

Comparing the time of manual test runs with the time taken to write a test script, we get the number of test runs after which the cost of manual testing exceeds the time taken by the automation of the same test case, i.e. the so-called break-even point. Each test run is significantly cheaper if the test is automated. For more information on the analysis phase of the results, see section 3.4.2. The research was primarily done in the short term, so there is no need to consider the time required to maintain the test scripts.

3.3.3 Exporting test coverage from SonarQube

Quantitative data is first and foremost collected in test coverage from various development tools. For this study, SonarQube was selected. The information gathered via SonarQube's web API includes the date and time in the format presented in the introduction section of chapter 3.3 and the test coverage for the whole codebase. The coverage value is represented in the percentage of lines covered. SonarQube also offers test coverage for added new lines, but this study focuses on the overall coverage. Test coverage is used as a metric of software quality. Comparison to other variables, such as the number of bugs found late in the release cycle, can reveal correlations. The resulting findings are discussed more in chapter 4.

3.3.4 Exporting customer issues from Jira

Each bug or development idea directly reported by the customer is marked as a customer issue in Jira. Even though not every customer issue means an error in the software's functionality, the number of said issues can be used as a metric of QA. A small number of found customer issues can be a sign of good QA and an adequate amount of testing before release. The high amount of customer issues usually means not enough testing or other assurance methods exist. This part of the study aims to collect every customer issue with a timestamp from the last three years and use that data to find correlations to improvements in QA, mainly the amount of testing. The result should be 1-3 graphs about both metrics to make it easy to conclude how the amount of testing impacts the number of bugs found by customers.

3.4 Analysis procedures

This chapter presents different ways to analyse collected data. The methods and formulas discussed in this chapter are based on proven practices for transforming raw data into useful information from previous studies. Some of these methods are used in business decision-making in corporations. Methods included in the study should primarily bring out the monetary value of TA, but the study also considers benefits that cannot be easily measured in money.

3.4.1 Return on investment (ROI)

ROI is generally calculated by dividing the net income by COI (Cost of investment) or investment gain by investment base. This work focuses on the former.

EQUATION 1

$$ROI = \frac{Total\ Benefits\ -\ Cost\ of Investment}{Cost\ of\ Investment}$$

To get the net income generated by automating tests, the formula subtracts the sum of all monetary benefits by the COI. What to include into the equation is not an easy task, given how many indirect benefits and significant savings can occur years after the initial test case was created. Likewise, chapter 2.1.3 discussed how most of the costs generated by automated tests are due to maintenance, which becomes a more considerable expenditure as the software gets older. However, the study will not focus on costs beyond a six-month timescale, so the maintenance costs aren't included in any analysis methods.

3.4.2 Break-even analysis

Break-even analysis can choose between multiple design options when the cost of those options is a function of one or more variables (Tockey 1997, p. 35). The purpose of Break-even analysis is to calculate the break-even point. Usually, this is done by dividing the fixed costs by the net return of a single unit. The study uses the cost of the initial implementation of the test case and divides it with the savings brought by single test execution. The formula ignores all other expenses related to TA, such as maintenance.

EQUATION 2

$$BEP\ = \frac{Initial\ cost\ of\ an\ automated\ test}{Savings\ per\ test\ execution}$$

To calculate the savings for single test execution, it's best to use manual testing as a reference point. Other savings related to TA are more indirect and less formulable to the equation. BEP calculated this way results in the number of test runs needed for the cost of manual testing to exceed the initial cost of automating the same test case.

4 RESULTS

The Results section presents the research results obtained through interviews and quantitative methods. There are two interviews, and the first of the subjects work at the university. The other interviewee works as a tester for the contracting company. The Test Coverage and customer issues section compare the data collected from both variables and attempts to find out if there is a correlation between the two. The Break-even point for the test execution section includes 13 test cases, which were initially manually executed, and the time it takes to automate them in minutes. These are marked in the table, which also includes the Break-even point, i.e. the required number of executions by hand, after which the automation pays for itself in terms of working hours.

4.1 Interviews

This chapter analyses the results of the interviews one by one. Questions and answers were in Finnish and had to be translated to English. The questions have almost direct translations, but the answers are not written word for word in the text. Instead, the topics discussed are brought up to be understood in both languages without losing critical information. Two interviews were held and analysed in the same order as they were conducted.

4.1.1 Interview 1

The first interview was pretty unstructured and only included four questions. The subject of the discussion was a professor at Tampere University's Information and communication technology department. The questions are numbered from 1 to 4 in chronological order. They are roughly translated into English. Under each question is a summary of the answers to the given question.

1. **How active is the research on software testing these days?**

   The interviewee thinks that the research work in software testing is reasonably active but not as busy as it used to be. The perspectives of the research change constantly. One way to find out how much

software testing has been brought up in recent studies is to find publicly available mentions from the latest ICSE conference papers.

2. What are, in your opinion, essential questions related to software testing?

One of the most critical questions is how the test suite is selected in the interviewee's opinion. Especially in large software systems, the testing work is starting to become a heavy process to upkeep. The interviewee said that an industry practitioner he was in contact with explained how running all of their tests took such a long time that even if the tests were set to run during the night, they were not finished when employees came back to work in the morning. Another hot topic is how cloud-native applications are tested. It's also important to note how cost-effectively the tests can be developed and maintained. Many testing tools exist to reduce the burden that testing brings. One of the latest and exciting ways to determine what to test is using artificial intelligence.

3. In what ways can the benefits of TA be measured?

The assumed benefits come from the increase in the quality of the software and the ability to execute test cases often. Automated tests are an essential part of the continuous integration method. Performing the test case often enough ensures that the quality of deployed code remains adequate and enables shipping the software multiple times per year. The interviewee thinks that automation saves time compared to manual work but doesn't believe TA is being developed solely to save money in the development process. He believes that the primary function of automated testing is to lower the quality costs of the SUT.

4. How much do companies use academic research, and what could be done to encourage companies to work together with academia?

"Since we are in university space, I'm obligated to say that the collaboration with software companies is inadequate", the interviewee says jokingly. However, diploma and master's theses are powerful mediums to create academia-industry collaboration in all seriousness. Recent research findings are usually brought to companies' knowledge through theses. Some collaboration projects between the university and software firms have been implemented. What it comes to increasing collaboration, the needs of the universities and the requirements of the companies and the expectations of financial

instruments are sometimes hard to coordinate. Thesis works are still the best way to increase collaboration. They often include research done by the student for the company. Another way is for firms to contact members of academia and propose collaboration projects directly.

Fifty-nine papers belonging to the 2020 conference and 60 documents belonging to the 2021 conference were found when searching ICSE conference papers using the keywords “tests” or “testing”. The 2020 conference included 142 articles and 151 in the 2021 conference. This means that around 40 % of the papers in both conferences were related to software testing in one way or another.

4.1.2 Interview 2

The second interview was held in the contracting company. The selected subject was a senior test engineer with a wide range of experience in different testing methods and tools for testing native applications. The questions were formulated based on the first interview, and the responses collected from the first interview were always presented to the interviewee before each question. In this way, the ideas raised by the university representative on the topics of the questions were brought to the attention of the second interviewee. The questions and their answers are listed in chronological order below and have been roughly translated into English.

1. **Software testing is under a lot of research, and many research articles and other publications are produced every year. Do you use industry research data in your daily work, and how?**

   The interviewee states that they do not take advantage of the latest scientific research on software testing from a practical point of view and have not spent much time reading studies. They had more interest in research publications in the early stage of their career but paid little attention to them in their current work.

2. **In a previous interview, I asked the interviewee what he thinks is the most critical question in software testing and, according to the interviewee, how the test set is selected. Another trendy topic is artificial intelligence in testing and aiding in selecting tests. What do you think of these, and how do these appear in your daily work?**

   In the interviewee’s opinion, tests themselves are not hard to run in parallel with modern technology if multiple instances of the SUT or critical part of it can be run simultaneously. More important is to pay attention to the complexity of the tests, especially with automated tests. More complex test cases cause more problems and maintenance work than simple tests. Tests involving database transactions,

such as client applications, are usually essential to cover with testing. The interviewee hasn't used artificial intelligence methods but has done fuzz testing to find unexpected errors that human testers don't discover.

3. **The next question was about how the benefits of test automation can be measured. According to the interviewee, the primary purpose of testing is to guarantee the program's quality is developed. The benefits of test automation are reflected in the fact that tests can be run frequently. Third, he mentioned testing as an integral part of ongoing integration. What more could a practitioner add to these answers?**

   The interviewee agrees that the primary purpose of TA should be to guarantee the quality of the software's release version. Automated tests should be run frequently for them to be beneficial. When tests are run daily, they can be updated as the SUT changes, ensuring the test set remains reliable until the shipment of production-ready software. Generating data is also easier with TA since the same code used for testing can also prepare the test data beforehand.

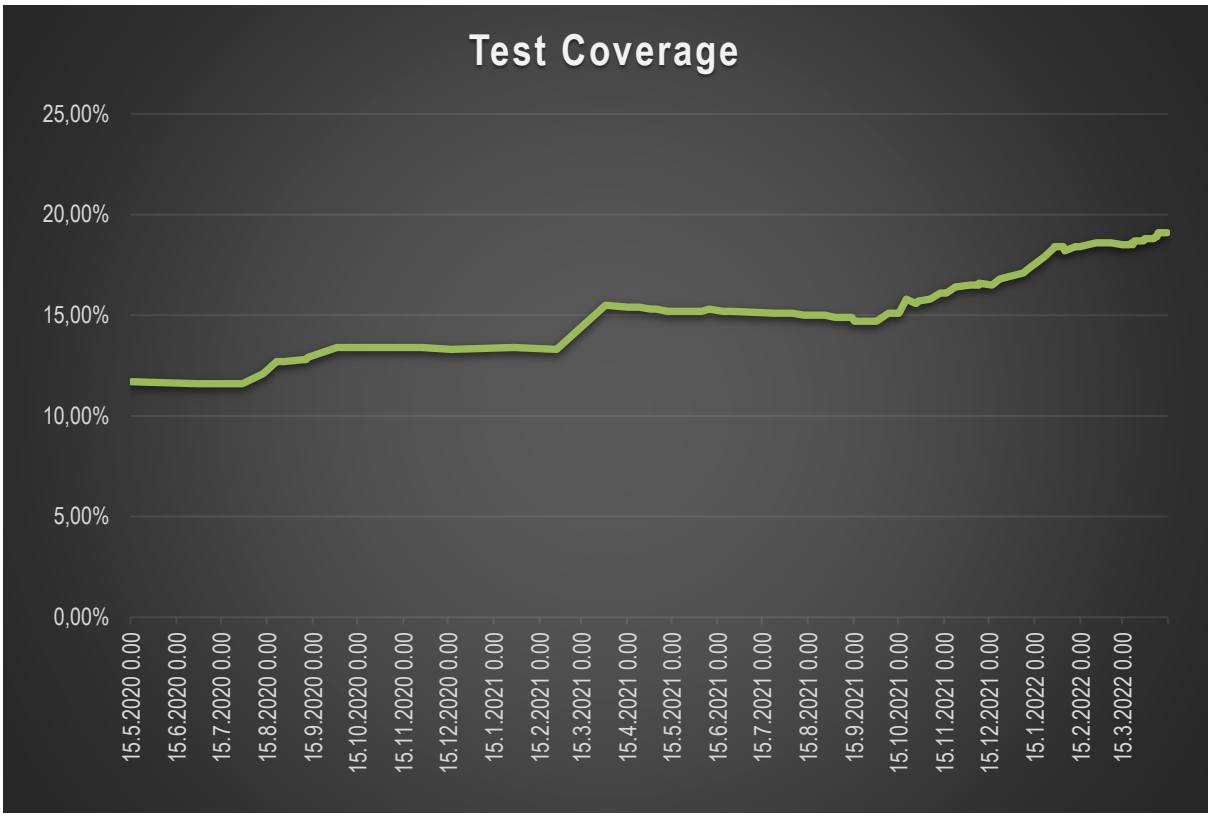
4. **Finally, the interviewee was asked how cooperation between universities and companies could be increased. In his opinion, theses are essential platforms for research in companies. Another way is to contact companies directly. How do you think cooperation could be increased, and what would you like from them?**

   The interviewee has received an offer to give a lecture to students in the past but has not taken part in any actual collaboration project

4.2 Test coverage and customer issues

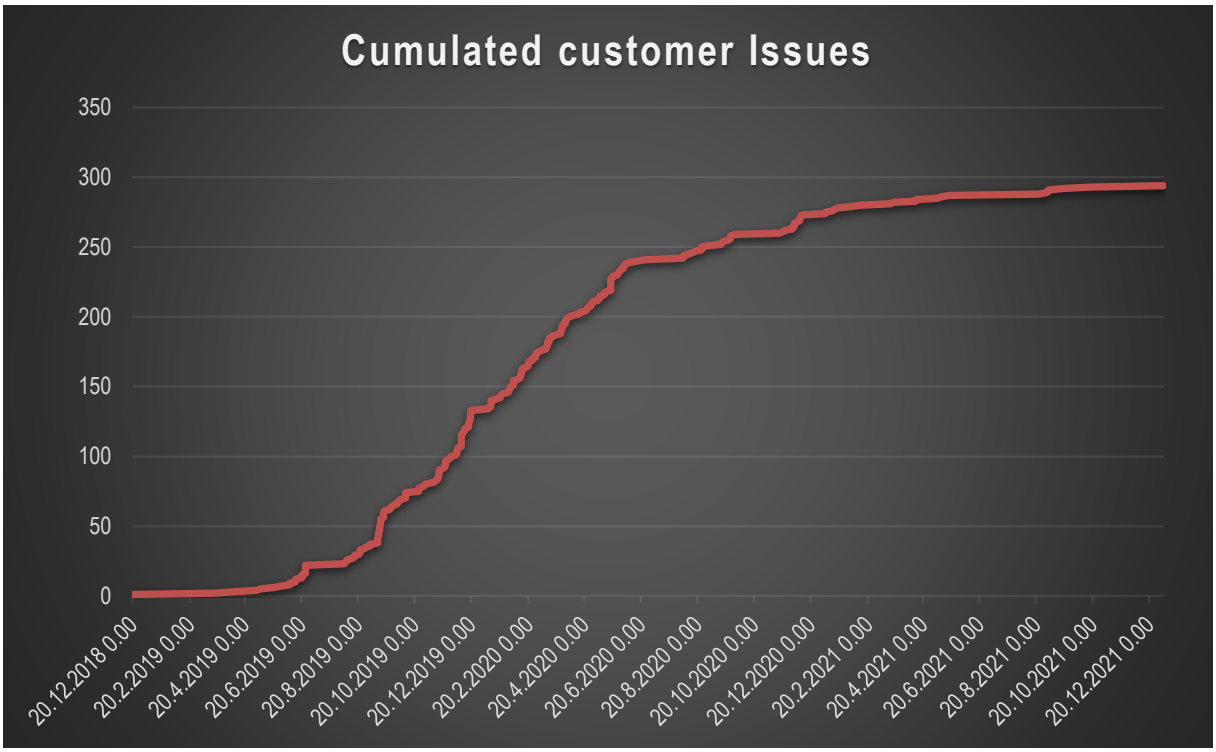
The chapter collects information on test coverage and customer issues and models the external correlation between the two. The pre-assumption is that as SUTs test coverage increases, customer issues, usually meaning bugs found by the customer, should decline. The test coverage data points include the timestamp and the percentage of the test coverage value corresponding to the time. customer issues are collected in the same way, including timestamps, but they are illustrated as the cumulative amount. Figure 2 shows the test coverage capacity in the period 16.4.2020 - 14.4.2022. Figure 3 shows the cumulative number of customer issues between 20.12.2018 and 16.2.2022. It is essential for the study's accuracy that the graphs are from approximately the same period. In this way, conclusions can be drawn from the change in one variable to the change in another variable.

FIGURE 2. Graph illustrating change in the overall test coverage over 700 days



Looking at Figure 2, it is good to remember that the coverage value is a ratio of the whole codebase. The percentage of test coverage would see a downward decline if new lines of code were added to version control without creating new test cases or expanding the existing ones. Therefore, even if the curve appears to flatline over some periods, the number of rows covered by the tests has increased.

FIGURE 3. The cumulative growth in customer Issoverd of 1111 days



Looking at both Figure 2 and Figure 3, conclusions can be drawn that after July 2020, the test coverage began to go up while the number of new customer issues started to decline. From 1.6.2019 to 31.5.2020, the average number of new customer issues per month was 19. From 1.6.2020 to 1.2.2022, the average was 3,5. At the same time, the test coverage rose from 11,60% to 18,40%. The number of new customer issues was lowest while the company invested in their TA. However, this does not mean the number declined solely because of testing. The company has put a lot of effort into improving its development and CI/CD pipeline processes. Testing is still an important part of continuous integration and cannot be ignored when looking at the different factors behind the increased quality of delivered software.

4.3 Break-even point for test executions

The calculation of the break-even point was presented in chapter 3.4.2. This section describes the execution times of manual test tickets collected from Jira and the time required to automate the same test cases. The thesis author creates an auto-automatic test. He has about one year of experience in system development and a few months of testing. Only those hours of work that directly contributed to the implementation of that test are counted as test writing time. Refactoring old code and other unrelated works were excluded from the calculations. The average execution and implementation times for all thirteen manual tests are shown in Table 3.

TABLE 3. Test execution and automation times for 13 manual regression tests

	Average execution time by hand (min)	Automation time (min)	Break-even point (manual executions)
Test 1	9	180	20,0
Test 2	8,5	215	25,3
Test 3	13	417	32,1
Test 4	12,5	382	30,6
Test 5	7,5	208	27,7
Test 6	44	681	15,5
Test 7	40	562	14,1
Test 8	67	733	10,9
Test 9	49	523	10,7
Test 10	15	109	7,3
Test 11	60	856	14,3
Test 12	7	120	17,1
Test 13	12,5	330	26,4
Average	26,5	408,9	19,4

As shown in Table 3, the Break-even point for automated tests is below 20 manual executions. During four increments (full calendar year), the maximum number of performances is usually 14-ns. This means the tests can be executed at least once every month. If the automated tests were run with the same frequency, it would take, on average, 16 months for them to break even. Computerised tests, however, can be run every single night if needed. This means they already catch up with the human tester in 14-15 days. Looking at the overall yearly costs alone for manual testing and automating the same test case, the cost of TA is on average 408,9 minutes or 6,815 hours, and manual testing can cost at most 15 * 26,5 = 397,5 minutes or 6,625 hours.

The results clearly show that the cost of automation tends to break even after a year, but what is more notable is the increase in test execution volume after the TA set is fully implemented. If the automation process were to occur at the start of the year, the test would be competent within one working day. Using 242 days' work year leaves 241 days for running the test case (weekends and holidays are omitted).

Kotaniemi (2021) reported in her blog post that the hourly rate for an experienced developer moving on both sides is 100 euros, and the hourly rate for a specialist is even closer to 200 euros. If 100 euros per hour is used as a reference cost for the software developer and tester, the initial implementation cost for one automated test is 6,815 * 100€ = 681.50€. In theory this would mean in total ((241 * 26,5) / 60) * 100€ = 10644.17€. If we calculate the ROI for one calendar year in this scenario, we get the following equation: (10644.17€ - 681.50€) / 681.50€ = 1461.87%. Executing a test case by hand every day is not practical, but it highlights how much more automation can improve the Quality Assurance of SUT when done right. Once the test case is automated, the tester no longer has to spend time testing that particular test case.

5 CONCLUSIONS AND FUTURE WORK

The study's findings show that automation can achieve certain advantages over manual testing, but in any case, it does not provide the desired additional benefit. It is advisable to select regressive test cases and perform them several times a year, for example, before each export to production. The study results also show that test automation can be used to perform test work that a human tester would not be able to perform, and tests can be performed at a much more intense pace. At best, all automated tests of the system can be performed every night.

Both interviews revealed the disconnect between the interests of members of academia and industry practitioners. One of the most important insights taken from the interviews was that manual and automated tests should be treated differently since automated tests can be run much more frequently and therefore cannot ideally compare to executing the test case by hand.

During the study, it was interesting to note how few new customer issues were generated after time was invested in improving test automation and increasing test coverage. The overall test coverage had increased by 63.2%. Most of the customer issues are bugs found in the customer's environment, so a significant decrease in them means that the bugs have not ended up in the version that goes to the customer.

Although this thesis does not study all the benefits of TA, it can act as a springboard for future experiments. For example, the thesis results can be used as reference values when creating a theoretical basis for a new case study, as was done in Section 2.1.3, Figure 1.

REFERENCES

Alégroth, E., Feldt, R. & Kolström, P. (2016). Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing. *Information and software technology*, 73, 66—80. DOI: 10.1016/j.infsof.2016.01.012

ANSI/IEEE Std. 730-1981. IEEE Standard Software Quality Assurance Plans. 345 East 47th Street, New York, NY 10017, Nov. 13, 1981. DOI: 10.1109/IEEESTD.1984.7435198.

Aranha, E. & Borba, P. (2009). Estimating Manual Test Execution Effort and Capacity Based On Execution Points. *International journal of computers & applications*, 31(3), 167—172. DOI: 10.1080/1206212X.2009.11441938

Baresi, Luciano & Pezzè, Mauro. (2006). An Introduction to Software Testing. *Electronic Notes in Theoretical Computer Science* 148 (2006) 89—111. DOI: 10.1016/j.entcs.2005.12.014

Barr, E. T., Harman, M., McMinn, P., Shahbaz, M. & Yoo, S. (2015). The Oracle Problem in Software Testing: A Survey*. IEEE transactions on software engineering,* 41(5), 507—525. DOI: 10.1109/TSE.2014.2372785

Bourque, P., Dupuis, R., Abran, A., Moore, J. & Tripp, L. (1999). The guide to the Software Engineering Body of Knowledge*. IEEE Software*, 16(6), 35—44. DOI: 10.1109/52.805471

Buckley, Fletcher J. & Poston, Robert. (1984). Software Quality Assurance. DOI: 10.1109/TSE.1984.5010196

Dobles, I., Martinez, A. & Quesada-Lopez, C. (2019). Comparing the effort and effectiveness of automated and manual tests. DOI: 10.23919/CISTI.2019.8760848

Dobles, I., Martinez, A. & Quesada-Lopez, C. (2019). Comparing the effort and effectiveness of automated and manual tests. DOI: 10.23919/CISTI.2019.8760848

Engineering research. (1969–2009*). International Journal of Software Engineering and Knowledge Engineering*, 23(9), 1343—1366. DOI: 10.1142/S0218194013500423

Garousi, V. & Mäntylä, M. V. (2016). When and what to automate in software testing? A multi-vocal literature review. *Information and software technology*, 76, 92—117. DOI: 10.1016/j.infsof.2016.04.015

Garousi, V. & Ruhe, G. (2013). A bibliometric/geographic assessment of 40 years of software

Garousi, V., & Felderer, M. (2017). Worlds apart: industrial and academic focus areas in software testing. *IEEE Software*, 34(5), 38—45. DOI: 10.1109/MS.2017.3641116

Garousi, V., Petersen, K. & Ozkan, B. (2016). Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. *Information and software technology*, 79, 106—127. DOI: 10.1016/j.infsof.2016.07.006

Glass, R. L. (2006). Software Conflict 2.0: The art and science of software engineering. “Developer.* Books”, 2006.

Grieskamp, W., Kicillof, N., Stobie, K. & Braberman, V. (2011). Model-based quality assurance of protocol documentation: Tools and methodology. *Software testing, verification & reliability*, 21(1), 55—71. DOI: 10.1002/stvr.427

Heusser, M. & Kulkarni, G. (2012). How to Reduce the Cost of Software Testing. DOI: 10.1201/b11258

Imtiaz, J., Sherin, S., Khan, M. U. & Iqbal, M. Z. (2019). A systematic literature review of test breakage prevention and repair techniques. *Information and software technology*, 113, 1—19. DOI: 10.1016/j.infsof.2019.05.001

Jayachandran, N. (2005). Understanding ROI Metrics for Software Test Automatio. Graduate Theses and Dissertations. https://scholarcommons.usf.edu/etd/2938.

Kaindl, H., Brinkkemper, S., Bubenko Jr, J. A., Farbey, B., Greenspan, S. J., Heitmeyer, C. L., Leite, J. C. S., Mead, N. R., Mylopoulos, J., Siddiqi, J., (2002). Requirements Engineering and Technology Transfer: Obstacles, Incentives and Improvement Agenda. *Requirements engineering*, 7(3), 113—123. DOI: 10.1007/s007660200008

Kaner C., Bach, J. & Pettichord, B. (2001). Lessons Learned in Software Testing: A Context-Driven Approach. *John Wiley & Sons*.

Koirala, S., & Sheikh, S. (2008). Software Testing: Interview Questions. *Jones & Bartlett Learning.*

Kotaniemi M. (2021). Ohjelmistokehittäjien palkkataso Suomessa 2021. Talented. https://talented.fi/fi/blog/ohjelmistokehittajien-palkkataso-suomessa-2021.

Kumar, D. & Mishra, K. (2016). The Impacts of Test Automation on Software's Cost, Quality and Time to Market. *Procedia computer science*, 79, 8—15. DOI: 10.1016/j.procs.2016.03.003

Lee, Jihyun & Kang, Sungwon & Kim, Chang-Ki. (2009). Software architecture evaluation methods based on cost benefit analysis and quantitative decision making. *Empir Software Eng* 14, 453—475 (2009). DOI: 10.1007/s10664-008-9094-4

Michael, J., Bossuyt, B. & Snyder, B. (2002). Metrics for measuring the effectiveness of software-testing tools. DOI: 10.1109/ISSRE.2002.1173225

Pančur, M. & Ciglarič, M. (2011). Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and software technology*, 53(6), 557—573. DOI: 10.1016/j.infsof.2011.02.002

R. Prest & R. Turvey. (1965). Cost-Benefit Analysis: A Survey. *The Economic Journal*. Volume 75, Issue 300. DOI: 10.2307/2229670

Ramler, R. & Wolfmaier, K. (2006). Economic perspectives in test automation: Balancing automated and manual testing with opportunity cost. DOI: 10.1145/1138929.1138946

Runeson, P. & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering: an international journal*, 14(2), 131—164. DOI: 10.1007/s10664-008-9102-8

Berner, S., Weber, R. & Keller, R. (2005). Observations and lessons learned from automated testing. *Proceedings. 27th International Conference on Software Engineering*, 2005. ICSE 2005., 2005, pp. 571—579, DOI: 10.1109/ICSE.2005.1553603

Smite, D. & van Solingen, R. (2016). What's the True Hourly Cost of Offshoring? *IEEE software*, 33(5), 60—70. DOI: 10.1109/MS.2015.82

Sneha, K. & Malle, G. M. (2017). Research on software testing techniques and software automation testing tools. DOI: 10.1109/ICECDS.2017.8389562

Solingen R, Berghout E. (1999). The goal/question/metric method. A practical guide for quality improvement of software development. McGraw-Hill

Strategic Planning and Economic Analysis Group. (2002). The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology*, 1.

Tockey, S. (1997). A missing link in software engineering. *IEEE software,* 14(6), 31—36. DOI: 10.1109/52.636594

Yin RK. (2003). Case study research. *Design and methods, 3rd edn. London, Sage*

Zhou, Z. Q., Tse, T. H. & Witheridge, M. (2021). Metamorphic Robustness Testing: Exposing Hidden Defects in Citation Statistics and Journal Impact Factors. *IEEE transactions on software engineering*, 47(6), 1164—1183. DOI: 10.1109/TSE.2019.2915065