LAPPEENRANTA-LAHTI UNIVERSITY OF TECHNOLOGY LUT
School of Engineering Science
Software Engineering and Digital Transformation

Master's Thesis

Ville Hartikainen

DEFINING SUITABLE TESTING LEVELS, METHODS AND PRACTICES FOR AN AGILE WEB APPLICATION PROJECT

Examiners: Prof. Jari Porras
Associate Professor Ari Happonen
Supervisors: Associate Professor Ari Happonen
M.Sc. (Tech.) Ilkka Toivanen

ABSTRACT

Lappeenranta-Lahti University of Technology
School of Engineering Science
Software Engineering and Digital Transformation

Ville Hartikainen

Defining suitable testing levels, methods and practices for an agile web application project

Master's Thesis 2020

84 pages, 22 figures, 1 table, 2 appendices

Examiners:	Prof. Jari Porras
	Associate Professor Ari Happonen
Supervisors:	Associate Professor Ari Happonen
	M.Sc. (Tech.) Ilkka Toivanen
Keywords:	web application, testing, testing definition

This thesis discusses how to define suitable testing levels, methods and practices for an agile web application project. Literature review, questionnaire and semi-structured interviews were selected as the research methods. The research is conducted in collaboration with the product creation services unit of Visma Consulting Oy. In the research, the factors that affect testing decisions in web application projects are identified and the suitability of different testing practices for different project contexts are modelled by investigating the benefits and drawbacks of the practices. The research concludes that project budget, criticality, schedule, personnel know-how and complexity especially affect testing considerations. In the definition of suitable testing practices, risk analysis and direction of the available resources to the critical parts of the application, are essential. The research highlights the definition of a testing plan, utilization of a wide range of testing methods and supportive practices. The results of the thesis can be utilized in the company's subsequent projects and the development of testing maturity.

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma

Ville Hartikainen

Sopivien testaustasojen, -menetelmien ja -käytänteiden määrittäminen ketterään web-sovellus-projektiin
Diplomityö

84 sivua, 22 kuvaa, 1 taulukko, 2 liitettä

Työn tarkastajat:	Professori Jari Porras
	Tutkijaopettaja Ari Happonen
Työn ohjaajat:	Tutkijaopettaja Ari Happonen
	DI Ilkka Toivanen
Hakusanat:	web-applikaatio, testaus, testauksen määrittäminen

Tässä työssä tutkittiin, kuinka määritetään sopivat testaustasot, -menetelmät ja -käytänteet ketterään web-sovellus-projektiin. Työn tutkimusmenetelminä käytettiin kirjallisuuskatsausta, kyselytutkimusta sekä puolistrukturoituja haastatteluja. Työ toteutettiin yhteistyössä Visma Consulting Oy:n tuotekehityspalveluyksikön kanssa. Tutkimuksen tuloksina tunnistettiin web-applikaatioprojektin testauksen määritykseen vaikuttavia tekijöitä sekä mallinnettiin eri testauskäytänteiden hyötyjen ja haasteiden kautta niiden soveltuvuutta tietyn tyyppisiin projektikonteksteihin. Työssä havaittiin projektin budjetin, aikataulun, kriittisyyden, henkilöstön osaamisen sekä kompleksisuuden vaikuttavan erityisesti testaukseen. Sopivien testausmenetelmien määrityksessä oleellista on arvioida projektin riskit ja keskittää käytettävissä olevat testausresurssit tärkeisiin kohteisiin. Työn tuloksina korostuu testaussuunnitelman laatiminen, laaja-alainen kehitysprosessiin integroitu testaus sekä testausta tukevien käytänteiden hyödyntäminen. Työn tuloksia voidaan hyödyntää yrityksen tulevissa projekteissa sekä testauskäytänteiden kehittämisessä.

ACKNOWLEDGEMENTS

Firstly, I would like to thank Ari Happonen and Ilkka Toivanen for brilliant guidance on my academic endeavours. I would also like to express my gratitude for the management of the PCS unit of Visma Consulting Oy for enabling me to conduct the research and to constantly learn more about software engineering. Many thanks to all the research participants for devoting their time and expertise. Last by not least, I would like to thank all the members of my family for the continuous support.

TABLE OF CONTENTS

1 INTRODUCTION 4

1.1 GOALS AND DELIMITATIONS 5

1.2 STRUCTURE OF THE THESIS 6

2 AGILE SOFTWARE DEVELOPMENT AND TESTING 7

2.1 OVERVIEW OF AGILE SOFTWARE DEVELOPMENT 7

2.2 TESTING IN AN AGILE ENVIRONMENT 9

2.3 AGILE TESTING LEVELS, ACTIVITIES AND SUPPORTING PRACTICES 15

2.3.1 Agile Testing Quadrant 15

2.3.2 Testing levels 17

2.3.3 Supportive practices 23

2.4 TESTING MATURITY LEVELS 25

3 WEB APPLICATION DEVELOPMENT AND TESTING 28

3.1 OVERVIEW OF WEB APPLICATIONS 28

3.2 WEB APPLICATION DEVELOPMENT 30

3.3 WEB APPLICATION TESTING 33

3.3.1 Testing levels 35

3.3.2 Performance-, load- and security testing 37

4 EMPIRICAL RESEARCH 39

4.1 RESEARCH METHODS AND BACKGROUND 39

4.1.1 Questionnaire 40

4.1.2 Semi-structured interviews 43

4.2 QUESTIONNAIRE ON PROJECT FACTORS THAT AFFECT TESTING DECISIONS 45

4.3 INTERVIEWS ON TESTING PRACTICES IN WEB APPLICATION PROJECTS 48

4.3.1 Significance of testing 48

4.3.2 Testing coverage in different levels 50

4.3.3 Automated and manual system testing 51

4.3.4 Non-functional testing *52*
4.3.5 Supportive practices *54*
4.4 INTERVIEWS ON PROJECT FACTORS IMPACT ON TESTING 56
4.4.1 Budget *56*
4.4.2 Criticality *56*
4.4.3 Schedule *57*
4.4.4 Know-how *57*
4.4.5 Technology *58*
4.5 OTHER THEMES THAT EMERGED DURING INTERVIEWS 59
4.5.1 Testing culture *59*
5 DISCUSSION 60
5.1 DEFINING TESTING PRACTICES FOR AN AGILE WEB APPLICATION PROJECT 60
5.1.1 Project factors affecting testing decisions *60*
5.1.2 Defining suitable testing practices for an agile web application project *61*
5.2 RELATION TO THE LITERATURE 62
5.3 SOFTWARE DEVELOPMENT PROCESS CONSEQUENCES 64
5.4 MANAGERIAL IMPLICATIONS 64
5.5 RESEARCH LIMITATIONS 65
5.6 FUTURE RESEARCH DIRECTIONS 65
6 CONCLUSION 67
7 REFERENCES 69
APPENDICES

LIST OF SYMBOLS AND ABBREVIATIONS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CSS	Cascading Style Sheets
DevOps	Development & Operations
DoD	Definition of Done
DOM	Document Object Model
E2E	End-to-end
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
OWASP	Open Web Application Security Project
SPA	Single Page Application
TDD	Test-Driven Development
UI	User Interface
XML	Extensible Markup Language
XP	Extreme Programming
XXS	Cross-site Scripting
XXE	XML External Entities

1 INTRODUCTION

In the domain of software product creation services and consulting, the projects handed out by customers, are diverse. It is a complex task to define a suitable testing level and supportive practices for each of them. The diversity of the projects is due to their characteristics, such as requirement complexity, estimated lifecycle and risks (Clarke et al. 2012). In addition, utilization of agile development practices introduces the challenge of integrating the testing activities to the iterative development process and shorter release cycles. Testing setup also controls to what extent the agile practices, such as continuous software development, can be utilized (Mäkinen et al. 2019). The domain of software testing is widely popular in academic research and there is a multitude of studies and publications on the theoretical background of software testing. The industry practitioners have also laid out models that outline and discuss the optimal testing setup (Cohn 2009; Fowler 2012; Mimick 2014). However, the modelling of suitable testing activities on project-basis in software consulting context remains quite unresearched.

The decisions on the agile software projects' testing setup are often based on expert knowledge and previous experiences (Drury-Grogan et al. 2017). The quality of these testing level decisions might result in under-testing or over-testing the software product. Both of which have consequences to the success of the project, former more critically, as undefined, unclear or insufficient testing scope might result in low-quality software product or extended project timeline. (Patton 2005) The success of the project from the business perspective revolves around delivering a fit product with enough quality within the scale of the budget. Therefore, it is of the essence to succeed in scaling the testing activities to a suitable level within the project and product context. (Black 2009)

Digitalization has transformed various industries. Existing operations are modernized and digitalized by using the latest technologies and mediums. (Kortelainen et al. 2017) Consequently, we are surrounded by web applications, some of which are handling critical business functions and sensitive user information, others require a high level of quality to

compete in the market. On the other hand, some web applications are less critical and benefit from rapid release to the market. Testing web applications is a difficult task due to their varying complexity and diverse features (Brandon 2008). Layered architecture and technological instability further convolute the development and testing activities (Kappel 2006). In such a context, consideration of the testing setup is paramount.

1.1 Goals and delimitations

The main objective of this thesis is to conduct academic research on how to select suitable software testing level and to identify methods and practices that support testing of an agile web application software project. To support achieving the research object, the following research questions will be answered:

1. Which project factors should be taken into account when considering the testing level of web application in an agile environment?
2. How to define sufficient testing level for web application projects relative to these project factors?

As an outcome of this thesis, the significant project factors affecting testing decisions in web application projects are identified and consideration of suitable testing activities relative to these project factors is produced. The research is conducted in collaboration with Visma Consulting Oy, more specifically with its Product Creation Services (PCS) unit. The unit is offering software product creation services in various fields and working on multiple diverse and often fast-paced agile software projects concurrently. Based on the information gathered during the research, a model is constructed, that supports the decision-making of testing activities in future web application development projects in Visma Consulting Oy. Academically, the thesis contributes to modelling and discussing current industrial practices and issues in the field of software testing.

1.2 Structure of the thesis

Section 2 outlines the high-level project context of the thesis, agile software development, and discusses the testing considerations and activities that are of the essence according to literature. In Section 3 an overview of the application context of the thesis, web application development, is given and the general testing considerations in the web application context are discussed. Section 4 presents the empirical research for the thesis. Section 5 is reserved for synthetization and discussion of the research results. Finally, in section 6, the research conclusions are presented.

2 AGILE SOFTWARE DEVELOPMENT AND TESTING

The following chapter discusses testing in agile software development context. The chapter outlines the testing levels, methods and practices that are depicted in literature. Also, testing maturity levels are discussed.

2.1 Overview of agile software development

Agile Manifesto (Agile Manifesto 2001), published in 2001 by a group of software industry figures, outlined the general values and principles of agile software development. In agile software development, the following core values are of the utmost importance:

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

To this day, a wide range of different agile software development methodologies and frameworks have emerged and evolved, such as Scrum, Kanban, Xtreme Programming (XP). All these different methodologies and frameworks cherish the agile values and therefore aim to focus on delivering valuable software to customers. (State of Agile 2019) The key principles of agile software development focus on team and customer collaboration, iterative development and shortening the release cycle. By these means, agile software development responds to change. (Agile Manifesto 2001)

For comparison, in traditional software development, there are clear, structured and documented phases for planning, designing, implementing, testing and deploying the software. Moving to the next phase requires the completion of the previous. If the project goals and customer needs are not clear at the beginning of the project, the traditional approaches might not work. In agile software development, these phases are completed in

iterations with short intervals (Figure 1). The iteration length often varies from one to four weeks. Iterativeness and continuous feedback cycle with the customer enable the project team to respond to changes. (Douglas 2016)

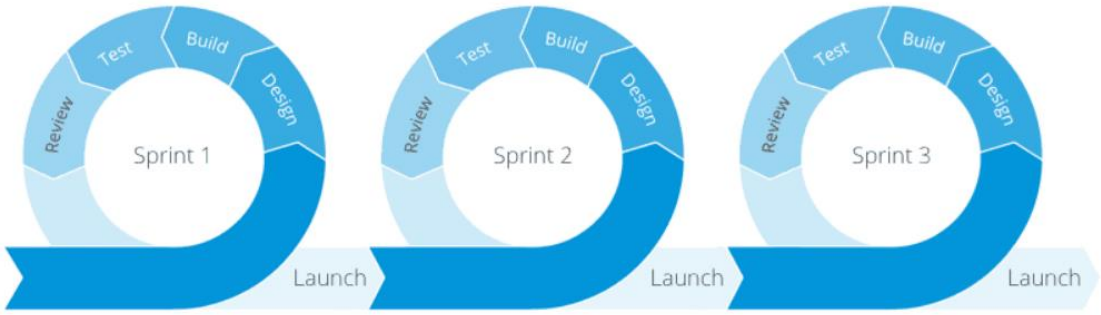

Figure 1 Agile development cycle (Goodman 2019)

According to the respondents of 13th Annual State of Agile survey (State of Agile 2019), agile software development is wildly popular in the software industry and continuously adopted by organizations. Only 4 % of the respondents did not have agile teams in their organization. Scrum-framework is currently the leading agile process framework in the industry. However, in practice, it is not uncommon to organize the day-to-day agile software development by combining activities from several methodologies and frameworks to so-called hybrid methodologies. By adopting agile methodologies and practices, teams are trying to accelerate software delivery, manage frequently changing requirements and increase productivity. The main benefits of developing software in an agile manner include the ability to manage changing priorities and to improve project visibility and business/IT alignment.

Iterativeness of agile software development has created a need for extensive automation of quality assurance and release pipelines to achieve high-quality and continuous workflow as well as releases. In DevOps (Development & Operations) methodology, development and operations, depicted in Figure 2, are integrated and exercised as a joint effort. (Kvhan 2017) Emphasis shall be put on automating development and testing activities as well as

configuration and environment management. To achieve such feats, practices such as continuous integration and deployment are embraced. By such means, higher-quality software is developed and released with ease. (Toivanen 2019)

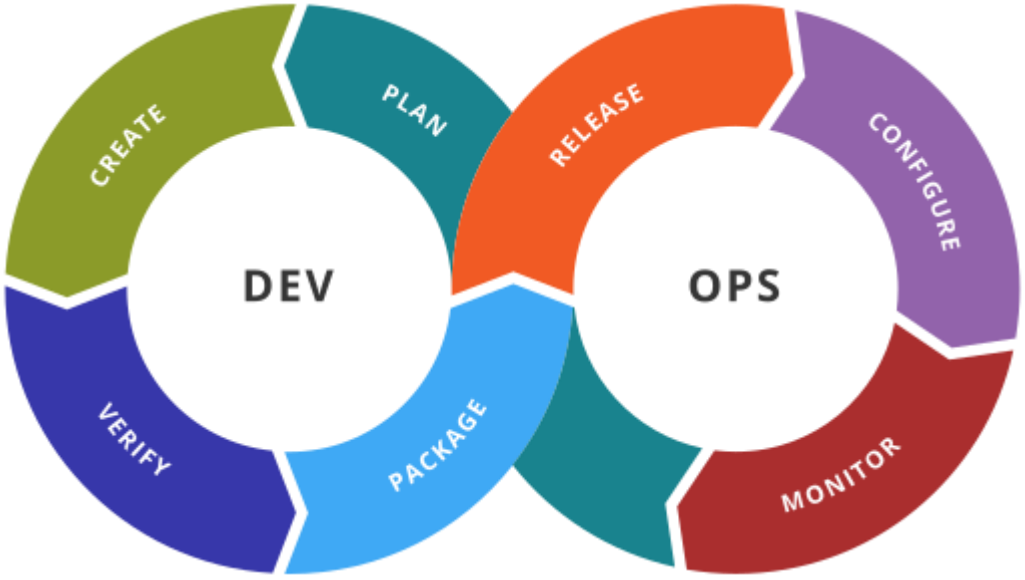

Figure 2 DevOps (Kvhan 2017)

2.2 Testing in an agile environment

Testing is an integral part of software quality assurance. Testing is an activity that aims to detect failures in the system's code or architecture. (Casteleyn et al. 2009, pp. 255-292) Planning of the testing level and completeness of the testing activities are driven by the initial risk assessment. Each software system has an acceptable level of quality, meaning that the software type and project context dictate the requirement for testing completeness. In general, testing activities aim to validate product quality and mitigate project risks. (Graham et al. 2008; Hambling 2010) The ISO (ISO/IEC 25010: 2011) software product quality characteristics (Figure 3) outlines the necessary aspects for quality evaluation and consequently guide the testing activities.

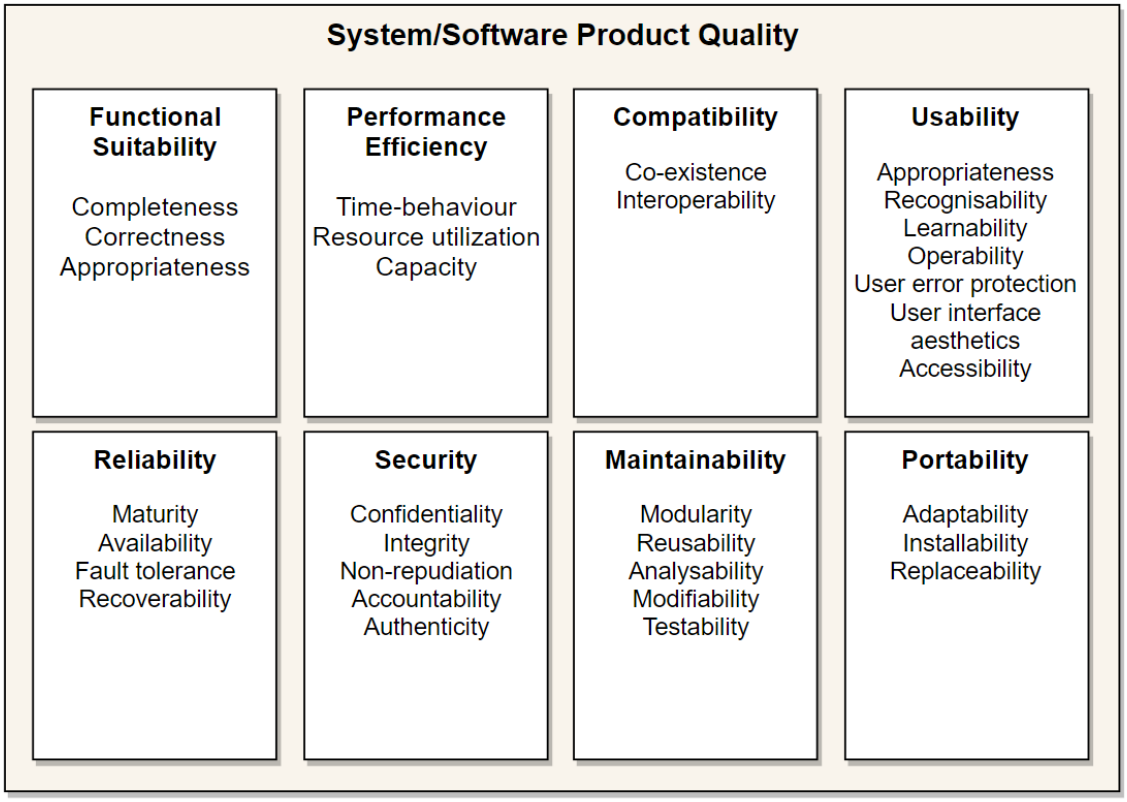

Figure 3 Software product quality characteristics (ISO/IEC 25010: 2011)

According to Patton (2005), there is an optimal testing effort for every software project (Figure 4). The aim of the project management viewpoint is to hit the optimal testing effort during the project execution. An additional layer of complexity is introduced by the fact that testing influences all conflicting areas in project-level; time, costs and quality (Kappel 2006, pp. 173). In Black's (2009) view, especially in agile software projects, the amount and rightness of features is another dimension that further convolutes the context of testing, as depicted in Figure 5.

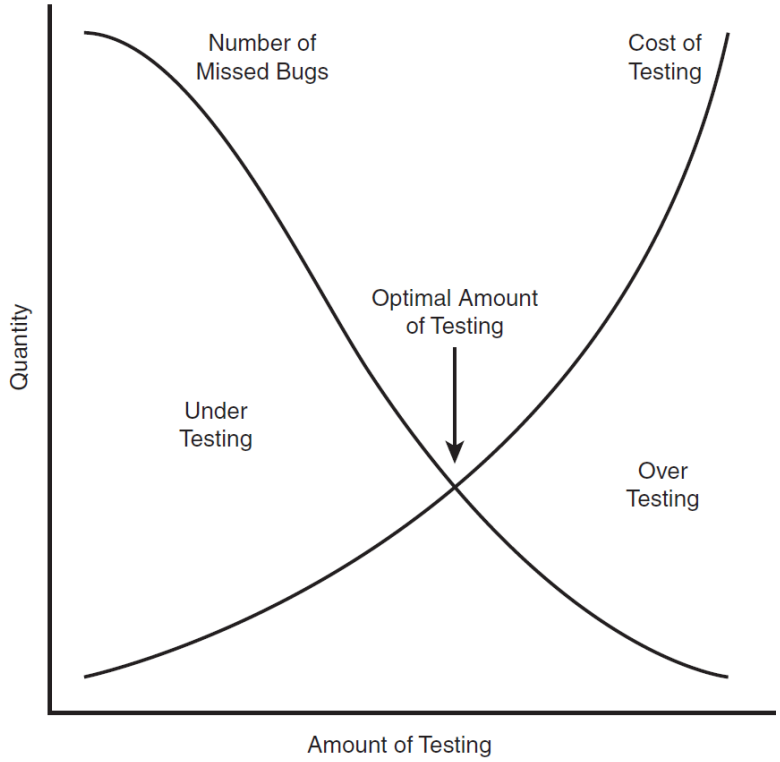

Figure 4 Software project test effort (Patton 2005, pp. 40)

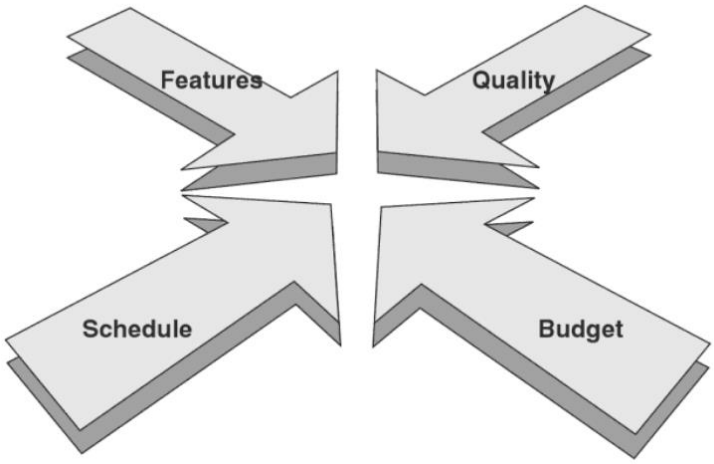

Figure 5 Project elements (Black 2009)

An organization can define its testing activities formally on a high level with testing policy and strategy as well as on project level via test plans (Kasurinen 2010). If testing policies and/or strategies are defined, they guide the definition of the project-specific plans

(Veenendaal 2019). In Kasurinen's (2010) study of multiple software organizations, it was concluded that two different approaches exist for test plan definition, design-based and risk-based approaches. In addition, changes to the testing process are often triggered by the need to correct problems instead of developing the process for quality and efficiency attributes.

Defining the testing objectives, scope, approach and focus of the testing activities is a necessity to enable the project team to deliver a high-quality product in the given timeframe. However, in agile development, the objective is not to deliver comprehensive and detailed test documentation. Instead, the focus should be on outlining and defining the necessary testing activities for the project in the project initialization phase. (Crispin et al. 2009, p. 86-88) The documentation for the testing activities at a high-level is viewed to be essential in an agile environment. The high-level testing plan should discuss the testing levels, types and quadrants that shall be exercised during the project execution. (Veenendaal 2019) Formulating such a testing plan is not easy. Context dependency and project unpredictability are key factors why the initial testing plan definition is a challenging process that requires judgment and skill. (Crispin et al. 2009, pp. 107) As such, the project execution should be monitored, to identify the possible need to change the initial testing approach (Veenendaal 2019).

Van Den Broek et al. (2014) research focused on testing in agile companies and proposed best practices for agile testing based on industry experiences. In the proposal, the first iteration of an agile software project should be allocated for preparation for the project. A testing plan should be formulated in conjunction with product characteristics and risks. Strategies for defect management, test automation and regression testing shall also be considered early on. Testing environments, as well as tooling, shall be put in place swiftly and early as possible to mitigate the possible risk of postponing the testing responsibilities, thus creating an unnecessary delay between the development and testing activities. Furthermore, the general recommendation is to include at least one tester per project to maintain product quality and to emphasize the customer perspective from inside the team. Discussion of such topics early in the project initialization phase is beneficial from the design

and coding standpoint, especially if the testing need for the load, performance, security, usability and reliability of the system are considered (Crispin et al. 2009, p. 18).

Automation is one of the key concepts in agile testing (Crispin et al. 2009; Fowler 2012). According to Van Den Broek et al. (2014) and Leotta et al. (2013) the decision to utilize or not to utilize automation should be made early on, as automation yields more value the longer it has been in place. This is emphasized in Figure 6 (Kyryk 2018). As the costs of implementing test automation are hefty and automation practice efficiency is application context-dependent, proper consideration on test automation utilization is necessary. As a general guideline, it is proposed that test automation should be considered when the project lifespan is at least 3 months. Test automation is also a test enabler, as for example load and stress testing of the system is possible only via automation (Crispin et al 2009, pp. 103, 283). In Kasurinen et al. (2010) research, it was discovered that only 26 % of the test cases in software development organizations are automated. However, according to the State of Testing survey (2019), three-quarters of the responders identify test automation & scripting as part of their job, which indicates that the automation practices are adopted more and more. Additionally, it was discovered that organizations are wildly different in regard to testing automation employment as depicted in Figure 7 – the majority of organizations have automated 10-50 % of the functional test cases.

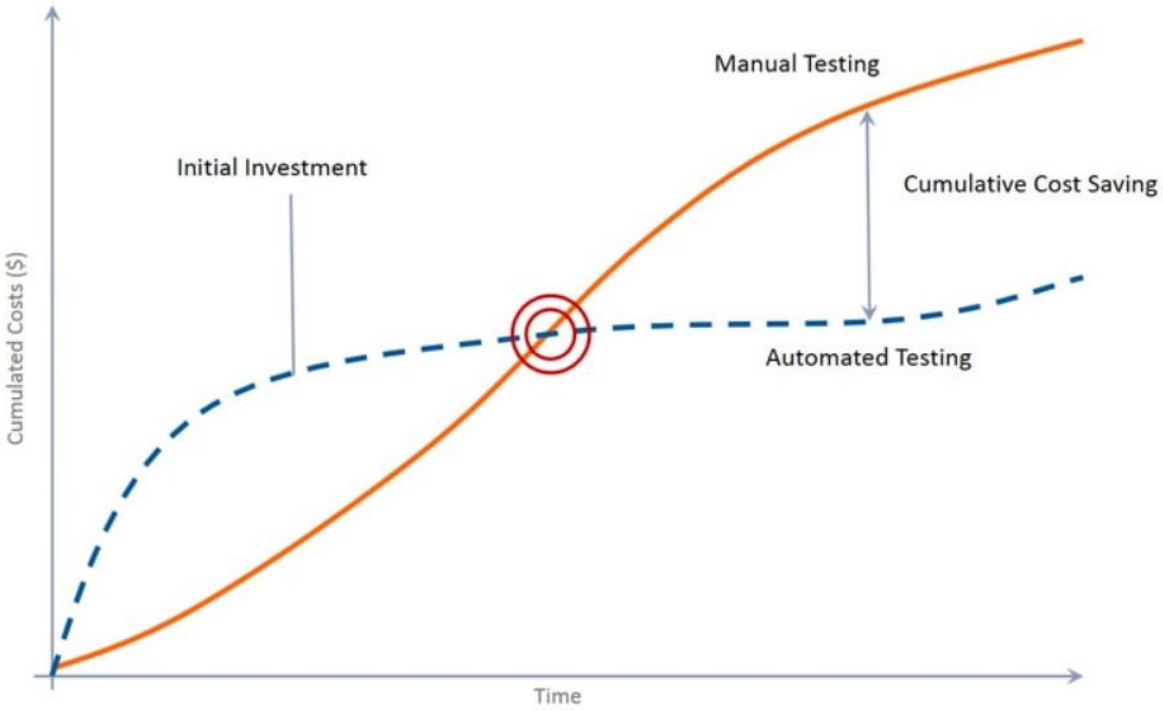

Figure 6 Time and cost of automated and manual testing (Kyryk 2018)

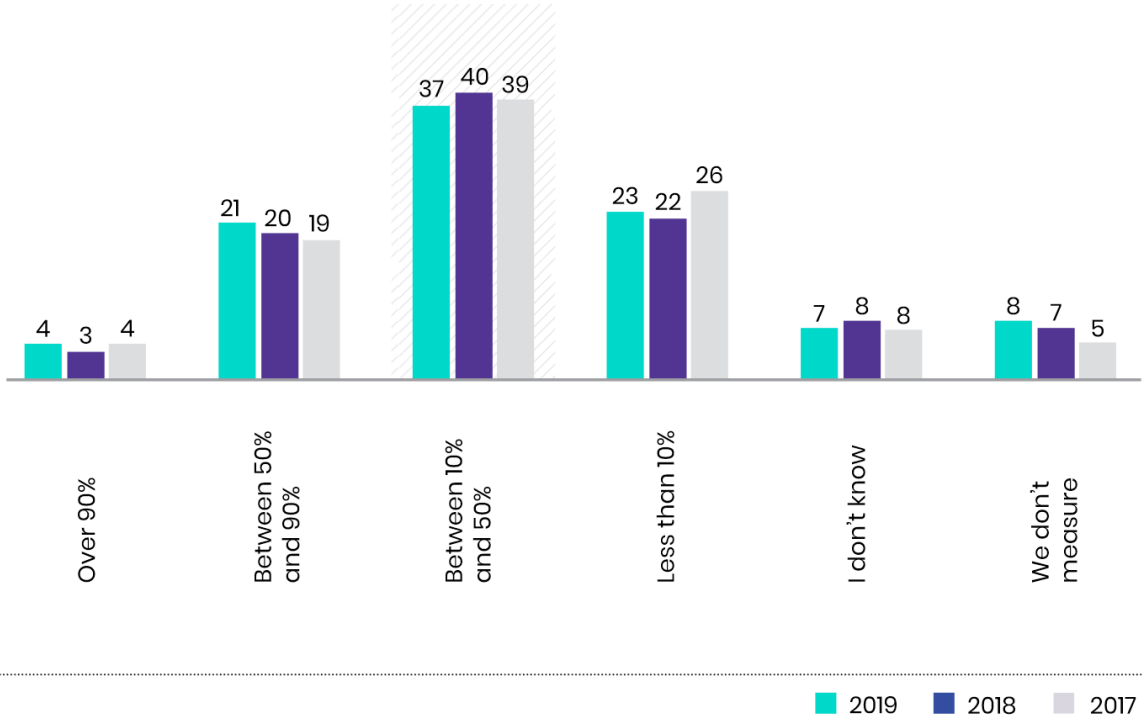

Figure 7 Test automation employment (State of Testing 2019)

In agile software development, maintaining the quality of the software product is continuous and collective in nature. Testing is the responsibility of the whole team; any project participant can act as a tester and complete tasks that have relevance to testing and product quality. (Crispin et al. 2009, p. 9-15) In today's software development, agile methodologies such as DevOps, further emphasize the necessity of implementing and exercising testing activities continuously and sharing the testing responsibility within the team. According to Veenendaal (2019), testing should be built into the iterations. Definition of Done (DoD) and the acceptance criteria for the individual features should discuss the necessary testing activities. These activities are usually derived from the high-level testing plan but depending on the type of the feature, it could be necessary to consider the testing activities from other viewpoints, such as non-functional requirements.

2.3 Agile testing levels, activities and supporting practices

In agile software development, delivering valuable software is the key concept. From the testing perspective, evaluating and extracting the value of the software is achieved by combining various testing methods and critiquing the product from different aspects. Automation holds great value in agile testing literature, but the role of manual testing should not be underestimated. To support the development and execution of an agile testing portfolio, the utilization of supportive practices such as continuous integration and test-driven development should be considered. (Crispin et al. 2009)

2.3.1 Agile Testing Quadrant

Agile testing quadrants (Figure 8) published by Crispin et al. (2009) divide the agile testing activities into four different sections. Each of the quadrants holds different types and levels of testing as well as enclose the supportive agile practices and methods. These quadrants can be used as a guideline and reference for testing activity definition on a project or feature level.

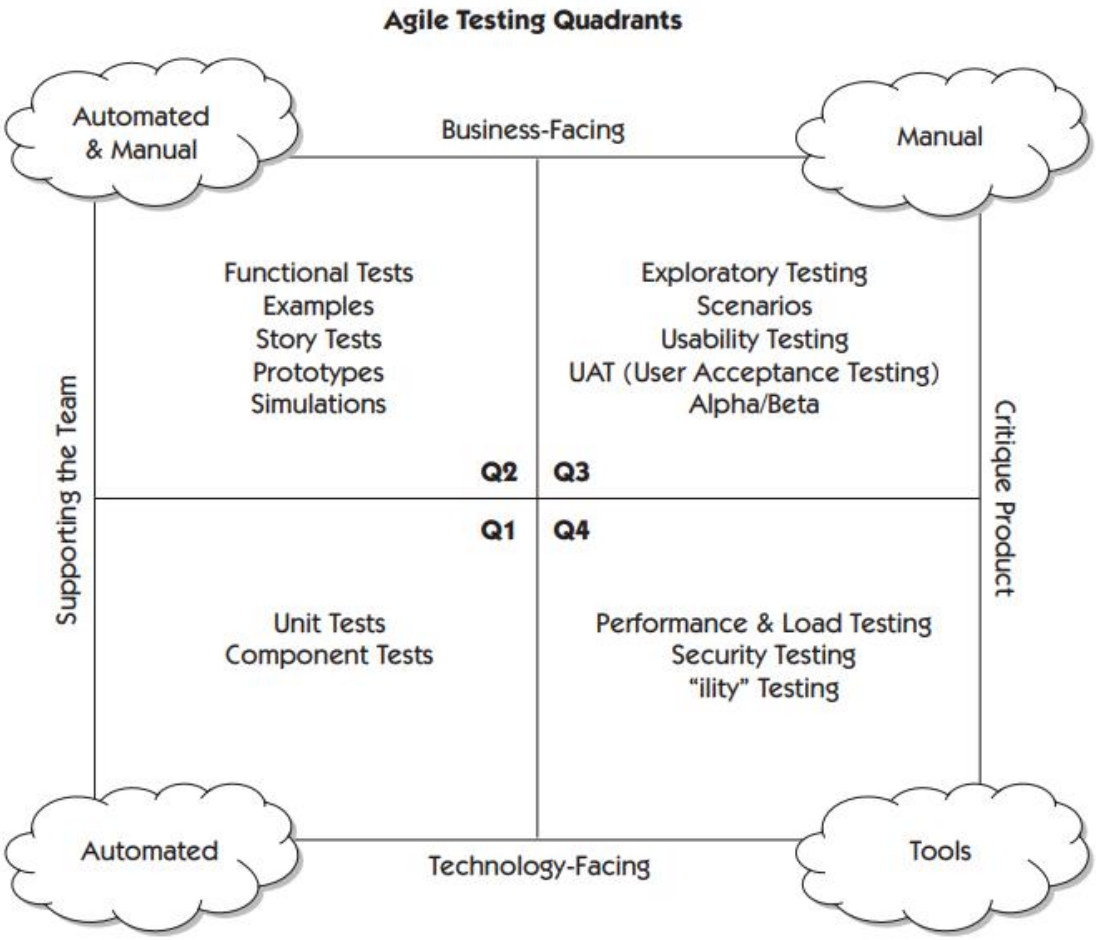

Figure 8 Agile Testing Quadrants (Crispin et al. 2009, pp. 98)

In the first quadrant, the focus is on developer-driven testing, including unit and component testing. Tests in quadrant one should be automated, and the practice of test-driven development should be emphasized. Testing in the first quadrant supports the testability of the system as a whole and rewards with higher code quality. In the second quadrant, the focus is on satisfying the user story acceptance criteria and general business conditions. This quadrant includes functional tests, examples, story tests, prototypes, and simulations. The testing activities in quadrant two drive the design of the system. Most of the technological testing in quadrant one and two have great potential for automation as they should be executed continuously to achieve quick feedback on the condition of the product. To enable this, the practice of continuous integration should be used. (Crispin et al. 2009, pp. 97-108)

Activities in the third quadrant are focused on manual testing that validates the acceptability of the feature or the system. The focus should be on exploratory testing and scenarios as well as usability. User acceptance and alpha/beta testing could be utilized depending on project type. The tests in the fourth quadrant are highly technical as they often are enabled by automation and require special expertise as well as suitable tools. (Crispin et al. 2009, pp. 97-108)

2.3.2 Testing levels

Generally, in software development, testing can and should be executed on various levels. There are usually three primitive testing levels depicted in literature; unit, integration and system testing (Chemuturi 2011, pp. 71-72; SWEBOK 2014). It must be noted that, in practice, the software testing terminology is convoluted and varies between practitioners and organizations (Fowler 2018; Vocke 2018). This was evident during the literature review. For example, the terms unit, integration and system level are in some contexts described as small, medium and large tests (Android Developers 2019). Tarlinder (2016) underlines the fact that even though the terminology might differ between practitioners and publications, the testing concepts and categorizations remain similar.

The test pyramid, originally introduced by Cohn (2009) and later revamped various software practitioners, is a representation of these testing levels and test quantity (Figure 9). By implementing testing on various levels, the confidence in the system and its quality is improved, debugging is made easier and the risk of introducing regression decreases, i.e. breaking existing functionality by introducing new (Cohn 2009, 311; Fowler 2012; Vocke 2018). The test pyramid was initially developed to guide test automation effort but later it has been expanded to represent and guide software testing effort in general. The general principle of the model is to focus on building the testing effort from the ground up and to emphasize automation. The activities described in the pyramid shall be automated. Unit tests form the foundation as they are fast to develop and execute. The middle layer consists of integration testing activities. The top layer consists of system-level testing. Lastly, all the

automated activities depicted by the pyramid should be accompanied by manual testing. The further up we go in the levels and activities, the more costly and slow the testing is. (Fowler 2012; Scott)

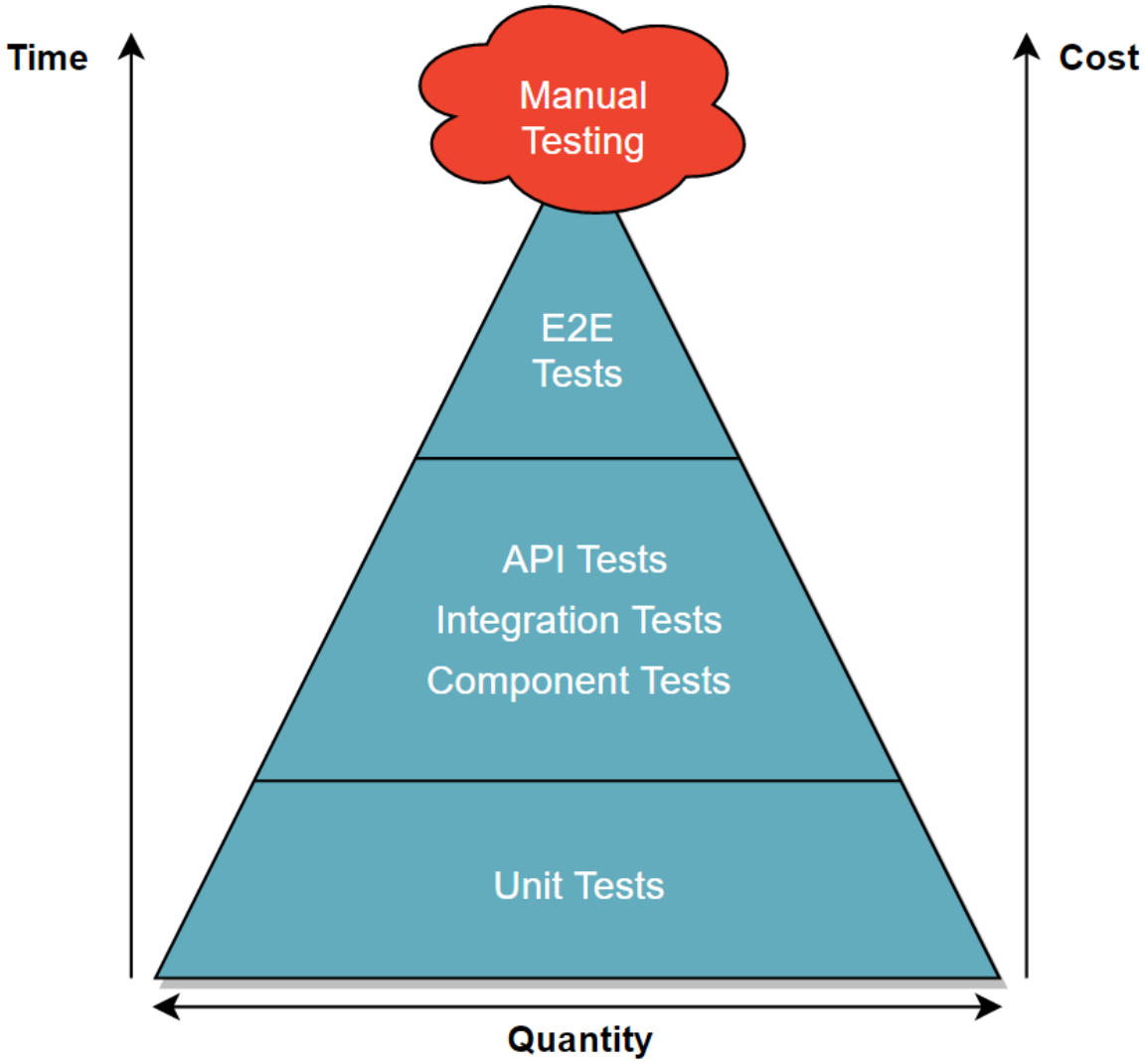

Figure 9 Testing pyramid (Cohn 2009; Fowler 2012; Vocke 2018)

The opposite of the test pyramid, testing ice-cream cone (Figure 10), is viewed as an anti-pattern that should be avoided. In this model, there are fewer unit and integration tests and emphasis are on automated and manual functional testing through the UI. Similarly, as in the test pyramid, cost and slowness increase as we move up in the figure. Consequently, in this model large quantity of tests are costly to implement and execute. (Fowler 2012; Scott)

Therefore the responsiveness, maintainability and reliability of the test setup are diminished (Vocke 2018). In practice, testing quantities at different levels vary on project and product basis and might not exactly follow either of the models depicted in literature and by software engineering practitioners (Contan et al. 2018).

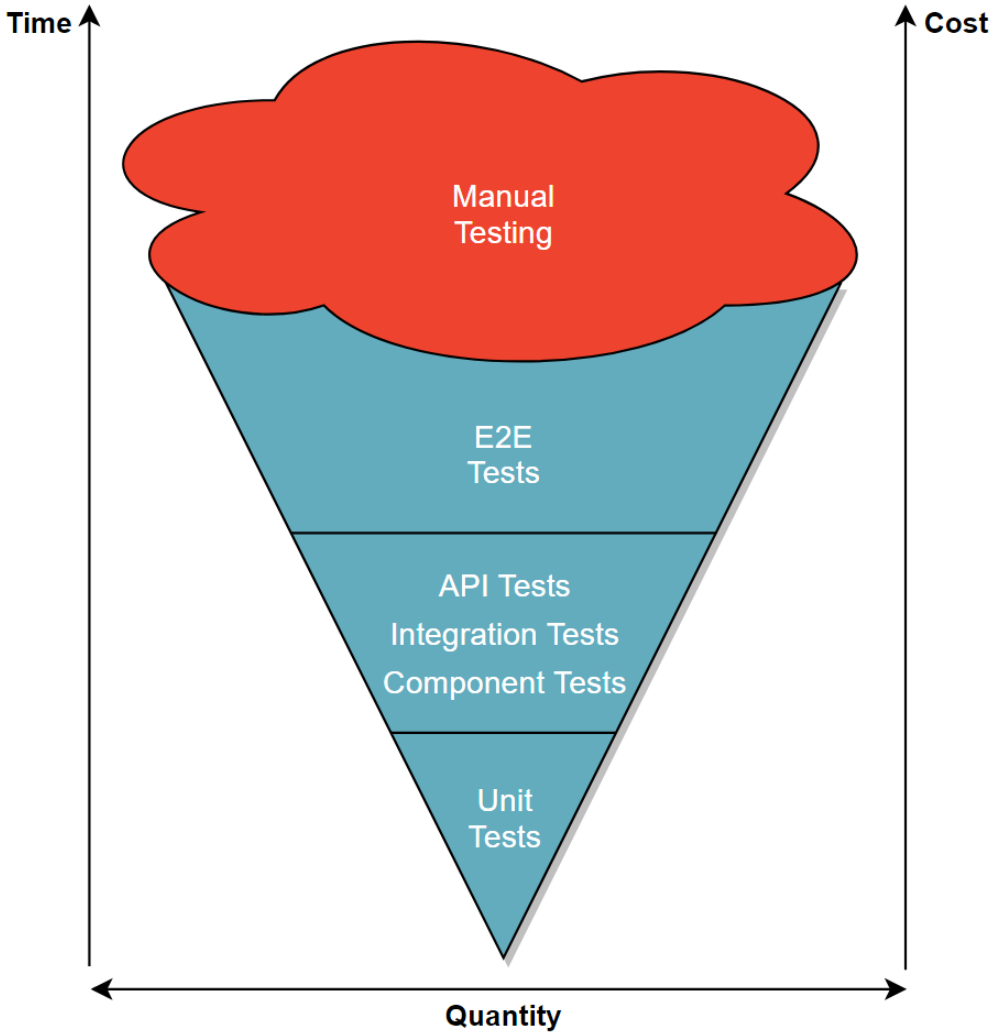

Figure 10 Inverted testing pyramid (Fowler 2012; Scott)

Developer written unit tests enable identifying faults in the earliest phase. Testing the smallest subsets of implementation constructs, such as functions, methods and classes, in isolation allows verifying their expected operation. According to Stack Overflow (2019) developer survey, less than half of organizations employ unit testing as part of their process

(Figure 11). In the integration layer, the testing range is wider as integrations happen on low-level as well as in high-level. In component testing, the proper integration of units to form larger entities, such as services and view components, are under scrutineering. (Crispin et al. 2009, pp. 109-127; Mark 2007) To verify specifically that the units and components are truly operating correctly, isolation is key. Isolation can be achieved by mocking the included dependencies, i.e. creating dummy implementations of the dependencies to avoid introducing side effects to the testable unit or component. (Mark 2007; Vocke 2018).

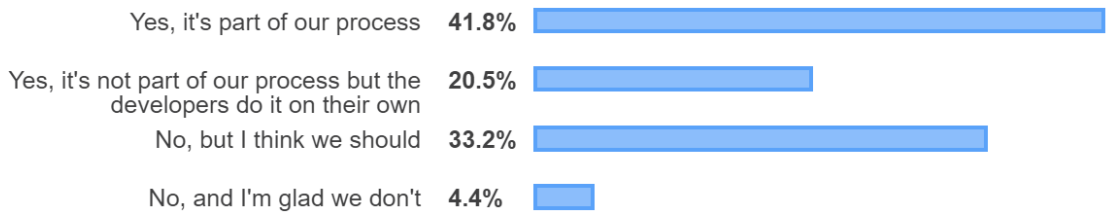

Figure 11 Answers to question "Does your company employ unit tests?" (Stack Overflow 2019)

Depending on the system, further integrations might be required. Integration testing refers to the validation of interoperation between parts, services and modules of the system (Fowler 2018; Vocke 2018). In Fowler's (2018) view, integration testing has broad and narrow scopes (Figure 12). In broad integration testing, multiple modules are active and in narrow integration testing, other modules are substituted by test doubles or mocks.

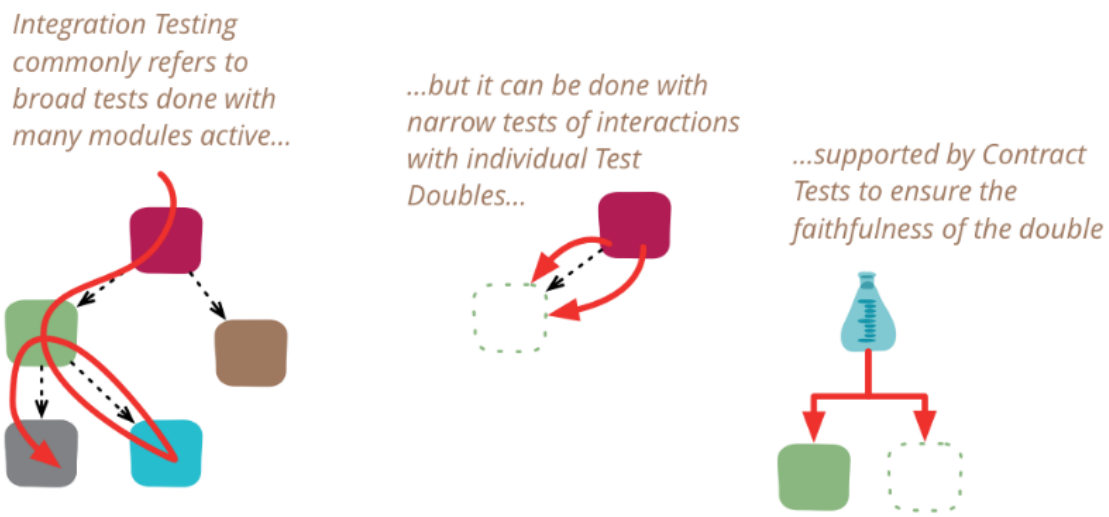

Figure 12 Integration testing scopes (Fowler 2018)

For example, usual integration test targets are the interaction with databases, external systems, larger modules and APIs (Application Programming Interface). Integration testing activities require further effort as the need for planning increases and tests require the implementation of test doubles and possibly the orchestration of live parts of the system, such as database (Fowler 2012). Regardless of the terminology interpretation, the integration layer of the pyramid is a widespread and important layer that validates the unit and component interaction and therefore limits the extent and load on which the system level end-to-end testing or manual testing is required (Vocke 2018).

Additionally, to verify the system operation as a whole, end-to-end (E2E) system testing is required (Tarlinder 2016, p. 34). In automated or manual end-to-end testing, the functionalities and usability of the system are verified by having all subsystems active, i.e. all systems integrated (Vocke 2018). Production databases should be cloned or emulated to mimic actual use-scenarios (Crispin et al. 2009, pp. 309). Automation of end-to-end tests is depicted to be difficult and time-consuming (Crispin et al. 2009; Vocke 2018). Incorporating all parts of the system, especially the GUI, to the testing, could result in fragile tests which are no use (Vocke 2018). Consequently, as we can see from the pyramid (Figure 9), the quantity of automated E2E testing should remain low. In end-to-end testing, the most

valuable user interactions should be mimicked (Vocke 2018). Additionally, consideration of whether to execute E2E-tests automatically or manually is necessary. The decision on this should be based on project risk and the coverage of unit and integration tests (Crispin et al. 2009, pp. 293).

Manual testing is a primitive testing type that enables defect detection and evaluation of the system's functionality as well as usability. For manual testing, there are two fundamental testing techniques described in the literature; test-case based and exploratory testing. In test-case based testing the manual testing is orchestrated by pre-designed and well-documented test-cases, i.e. scenarios. Therefore, the test execution is an easily reproducible and mechanical task. Exploratory testing focuses on experimentation and learning instead and do not emphasize test case documentation. In exploratory testing system is tested freely and possible inconsistencies are followed and reported. (Itkonen et al. 2014) Usually, a certain theme for testing is selected or testing is executed from the viewpoint of different user roles. (Crispin et al. 2009, pp. 201-202) Exploratory testing is viewed as a more suitable manual testing method for agile projects as it fits situations where product documentation is scarce, features are changing rapidly, and the project is time-limited. The effectiveness of both methods has been researched and there is no clear evidence of either being more effective in detecting defects. However, due to its lightweightness, exploratory testing fits into a wide range of projects and is proved to be efficient. (Afzal 2015; Itkonen et al. 2014) In the software testing field, manual exploratory testing is viewed to be a time-effective and cost-effective way to test a system. However, its effectiveness to detect regression is questionable. (Ghazi et al. 2015)

According to Crispin et al. (2009, pp. 217-239), depending on the system, further validation of its robustness could be beneficial. Performance, load and security testing enables to test whether the system and its design are sound enough to fulfil the non-functional requirements. Additionally, testing of the following aspects could be beneficial in some application domains:

- Maintainability
- Interoperability
- Compatibility
- Reliability
- Installability

2.3.3 Supportive practices

Along with test automation, continuous integration and test-driven development are key activities that support agile testing (Crispin et al. 2009). In continuous integration (CI), the codebase is inspected, built and automated tests are executed continuously in build server on every new merge to the mainline (Figure 13). These actions are taken to ensure that the integration routine is executed at the build server instead of only at the developer machine. Additionally, CI ensures that build stays green, i.e. possible faults are identified automatically by the integration routine and the proposed code merge is rejected until the identified issues are fixed. The effectiveness of continuous integration is based on the quality of the integration routine. Comprehensive and rapidly executing test routine enables the development team to efficiently get notified of integration faults, defects and regression that might be introduced by new changes. (Fowler 2006; iClerisy 2019; Meyer 2014; Mårtensson et al. 2017; Ståhl et al. 2013) Furthermore, continuous integration enables continuous deployment (CD) as the builds that pass continuous integration routines are ready to be passed for the deployment pipeline. In continuous deployment, the target is to achieve automatic releases to the production environment. Continuous integration and deployment have been adopted by software practitioners as according to State of Testing (2019) survey, 81 % of the respondents have employed some level of CI/CD-practices in their projects.

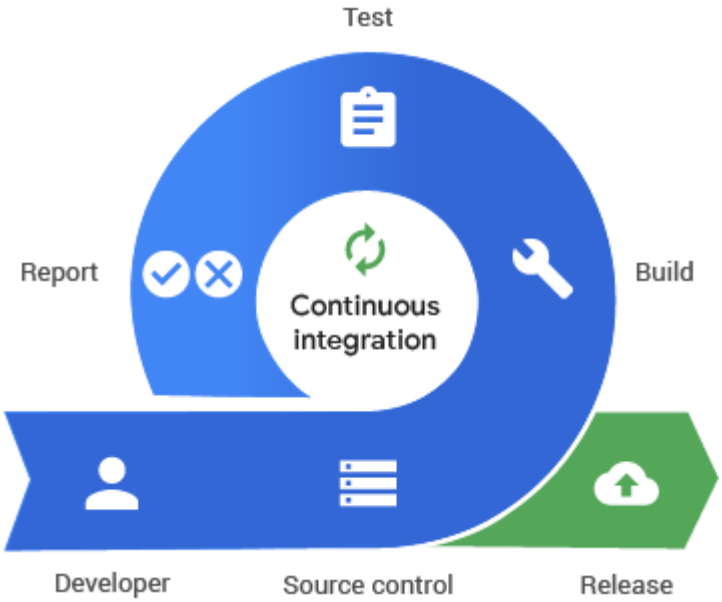

Figure 13 Continuous integration routine (iClerisy 2019)

The technique of test-driven development (TDD) can be utilized to support the implementation of a comprehensive test suite. The purest concept of TDD defines that unit and component tests should be designed and written before the implementation code. Additionally, high testing coverage is emphasized. The usage of TDD is stated to result in higher code quality and lower project cost. Although, there is evidence that its impact on software development is debatable. (Borle et al. 2018, Karac et al. 2018, Mark 2007) This is mostly due to the context-dependency, as in the real world, the tasks, application setting, and developer skill vary (Causevic et al. 2012; Karac et al. 2018). For developers, the adoption of TDD is experienced as difficult and chasing high coverage numbers might not better the quality of the product (Mark 2007). Additionally, TDD might slow the development and its adoption could be limited by not having clear design or requirements (Causevic et al. 2012). Approaching legacy code with the practice of TDD is also found out to be difficult (Causevic et al. 2011; Mark 2007) In practice, the benefits of TDD include confidence in system design as the features are planned more thoughtfully to enable designing and writing tests prior implementation. In addition, refactoring and changing code is more straight-forward as the existing test suite can verify correct operation after changes. (Mark 2007)

2.4 Testing maturity levels

Various models for testing maturity levels have been laid out by the software engineering community. Such models usually discuss the maturity of testing at five different levels. Ammann et al. (2016) discussed the testing maturity in layman terms by referencing Beizer (1990):

- There is no difference between testing and debugging
- The purpose of testing is to show correctness
- The purpose of testing is to show that the software does not work
- The purpose of testing is not to prove anything specific, but to reduce the risk of using the software
- Testing is a mental discipline that helps all IT professionals develop higher-quality software

In TMMi Foundation's (2019) testing maturity model, the levels are called initial, managed, defined, measured and optimisation. The model is process-oriented and as such more traditional but the concepts are applicable to agile development (Veenendaal 2019). At the initial-level, the testing process is unmanaged. To achieve managed-level, the testing policies should be in place. In defined-level, the organization should have defined testing standards and procedures to enable utilization of common practices in all of the projects. Also, non-functional testing aspects are required to be considered. At measured-level, measurement should be utilized to minimize defects. Lastly, the optimization-level requires advanced utilization of the measurement to enhance the testing process. (TMMi Foundation 2019)

Recently, due to the emergence of the agile continuous software engineering practices, the maturity of such operations has been modelled by a multitude of industry practitioners and organizations. In continuous deployment maturity models, authored by Rehn et al. (2013) and Mimick (2014), various aspects of continuous deployment are depicted. These models

are suited for identifying the current state of operations and support in the feat to advance to the next level. In Mimick's (2014) model the following five maturity levels are described:

- Base
- Beginner
- Intermediate
- Advanced
- Extreme

Additionally, such maturity model categorizes continuous deployment into four components:

- Building
- **Testing**
- Deploying
- Reporting

The maturity levels of testing are depicted in Figure 14. In base level, the first steps towards automated testing are taken by implementing some unit testing. The majority of the testing activities still remain manual. At the beginner level, some of the integration tests are automated, shifting the testing effort towards automation and the test portfolio consists of fast tests. In the intermediate level, system testing effort swifts towards automation as critical user paths are automated. At the advanced level, the test portfolio consists of automated tests and is supplemented only by risk-based exploratory testing. Additionally, the non-functional aspects of the system, such as performance and security, are validated by automated testing. In such setup, the critical paths of the system are covered by automation and acceptance testing, and consequently, continuous releases, are a breeze. Extreme level shares the same characteristics as advanced level but emphasizes even more extreme testing coverage and generation of usable information on expected business results. (Rehn et al. 2013; Minick 2014)

Generally, in testing maturity models, the overall testing coverage and the efficiency to detect regression increases the more advanced the level. Consequently, the lead time to new release decreases at each level as a more comprehensive test portfolio is executed automatically and efficiently (Mäkinen et al. 2019). Such feats improve overall product quality and contribute to the more efficient release process and continuous deployment pipeline. From the models, it can be observed, that non-functional testing activities are introduced at the more advanced levels.

Currently, according to Mimick (2014), the base level maturity is an industry-standard and the intermediate level is the targeted level of operations in most software projects. As to discuss the testing quantity models relative to the maturity model, the base layers go hand-in-hand with the ice-cream cone model and the levels in the extreme-end follow the pyramid model. As the former highlights manual testing and the latter encourages automation.

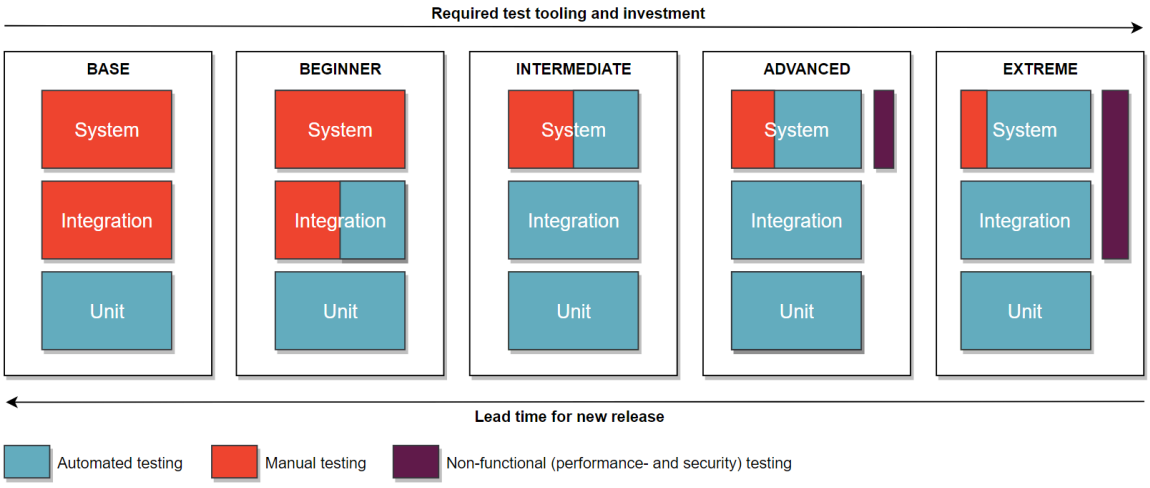

Figure 14 Testing maturity levels (Rehn et al. 2013; Minick 2014; Mäkinen et al. 2019)

3 WEB APPLICATION DEVELOPMENT AND TESTING

The following chapter outlines the general structure of web applications and the diversity of the development processes and technologies. Furthermore, the literature on testing of web applications is summarized.

3.1 Overview of web applications

Web applications are vastly utilized and complex systems that differ in functionality, scale and characteristics. However, all web applications are accessed through the web browser. (Brandon 2008, pp. 5; Kappel 2006 pp. 2-3) This is emphasized in the following definition of a web application by Kappel (2006, pp. 2):

"A Web application is a software system based on technologies and standards of the World Wide Web Consortium (W3C) that provides Web specific resources such as content and services through a user interface, the Web browser"

Web applications share the concept of client-server architecture and typically consist of three logically separated layers (Figure 15): presentation, application and data layers (Kappel 2006, pp.73-74; Laine et al. 2011). The presentation layer defines what kind of views are displayed in the browser and controls how the users can interact with the server via the HTTP-protocol (Hypertext Transfer Protocol). The application layer contains all the business logic for the system to function. It handles the HTTP-requests initiated from the clients' browser and queries the data layer to retrieve or store necessary information. The data layer consists of the database(s), tables, views and the data access functionalities as well as possible database logic or value manipulation with procedures and triggers. (JReport 2019; Mok et al. 2013)

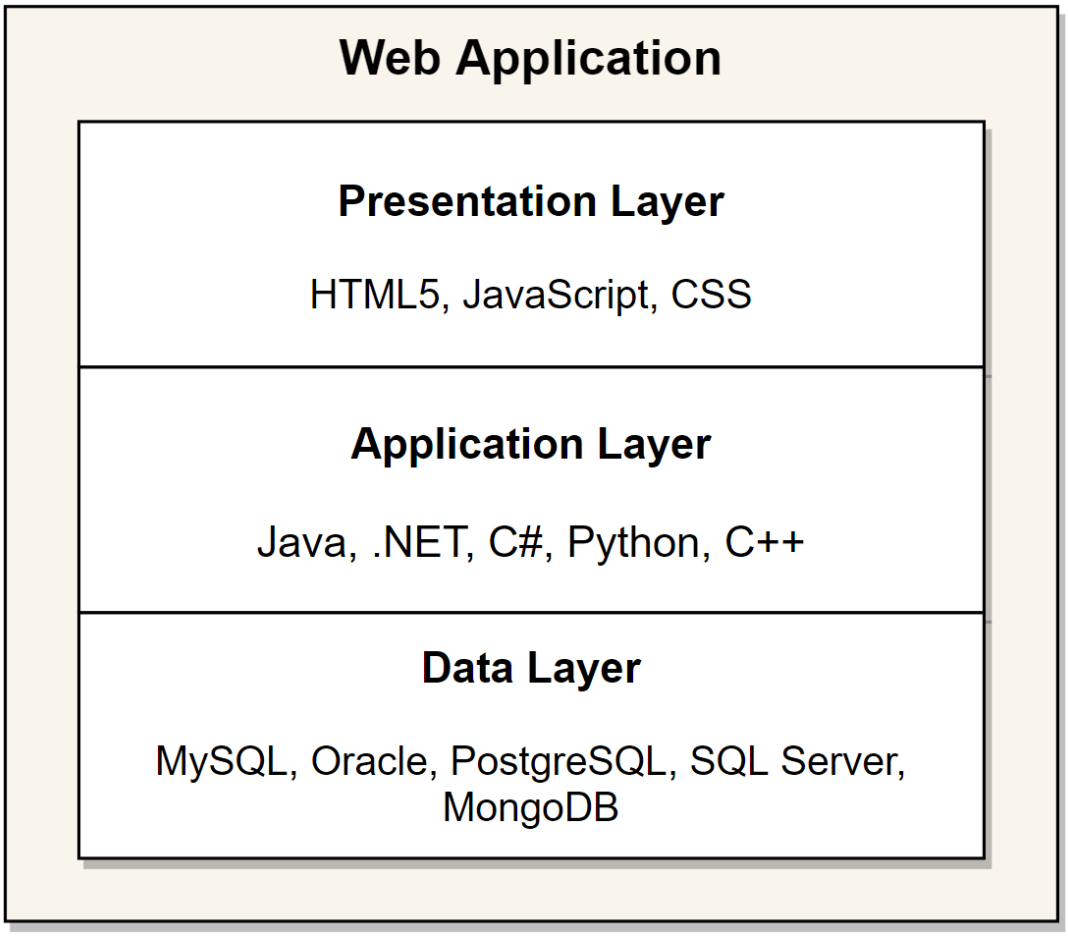

Figure 15 Web application layers and examples of technologies (JReport 2019)

Web application shares various characteristics that make the development of these applications difficult and different from other fields of software development. Web pages can show static and dynamic content in many forms such as text, graphics, audio and video. Web applications are often targeted for large userbases and the users are using the applications on various screen sizes in varying networks. Data intensity of the applications, i.e. content and database-driven nature of the system introduces also introduces concerns on security and privacy aspects. (Murugesan 2008; Arora et al. 2012)

3.2 Web application development

Web application development is characterized by involving an abundance of programming languages, concepts and frameworks. (Murugesan 2008; Casteleyn et al. 2009; Doyle et al. 2017) It is also typical to utilize existing libraries and tools to speed up the development and to avoid re-writing solutions to already solved issues, i.e. reinventing the wheel. (Kaluza et al. 2019) The wide range of these libraries and tools are open-source and therefore extremely accessible. (Alenezi et al. 2016; Vemula 2017) The shift pace at which these languages, libraries and frameworks evolve, is also one of the key characteristics of web application development. Due to the constant and rapid changes in the technological foundation, web application development emphasizes the knowledge and experience of individuals instead of standardized practices (Brandon 2008, pp. 5-7).

The trends in the tooling interest and adoption change year-by-year and some of the frameworks are more versatile and easier to work with than others as depicted in Figure 16. (Stack Overflow 2019) These rapid and possibly unexpected changes in the adoption and support for the specific framework could complicate the development and maintenance processes. On the other hand, this rapid development and open-sourcing of the development tools have made the creation of innovative solutions with web applications more accessible (Vemula 2017). Identification of beneficial and suitable tools from the sea of options is difficult (Kappel 2006, pp. 176; Kaluza et al. 2019).

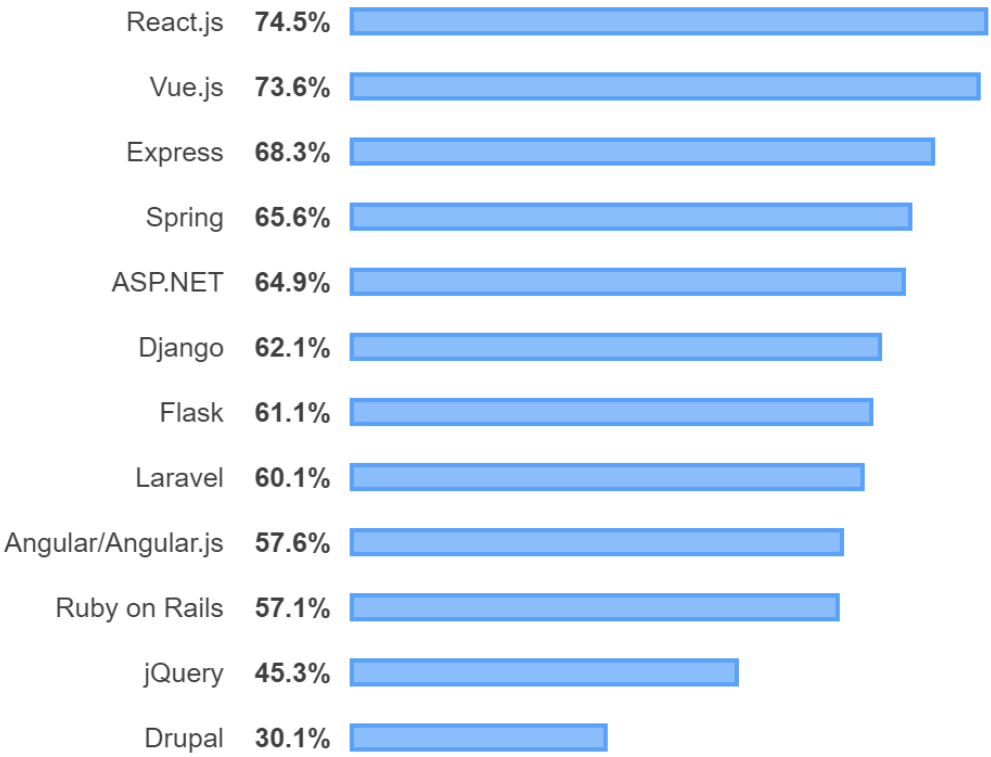

Figure 16 Percentage of developers that are currently working and expressed interest to work with the web framework in the future (Stack Overflow 2019)

To develop the presentational layer of the web application for the client browser, e.g. front-end develop, HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are involved in structuring and styling the page templates. Dynamicity and underlying logic are introduced to the web pages with JavaScript scripting language. (JReport 2019) In modern-day web application development, the presentational layer is often developed by utilizing CSS-frameworks and JavaScript libraries/frameworks. Several open-source CSS-frameworks exist, such as Bootstrap, Foundation and Materialize CSS. Additionally, often CSS pre-processor, such as Sass or Less, is used to extend the basic CSS-functionalities. (State of CSS 2019) The most popular JavaScript front-end frameworks/libraries currently include React, Angular and Vue.js (Hlebowitsh 2019). These front-end frameworks enable and facilitate the creation of single-page applications (SPA). In single-page applications, the page reloads are minimized as the application state and logic stored in the browser. The state is manipulated by executing asynchronous AJAX (Asynchronous JavaScript and XML)

API-calls under the hood and state changes are reflected dynamically to the web page with client-side rendering. With server-side rendering, dynamic web page content is constructed in the server, based on user navigation and input. In today's web development, client-side rendering is utilized in the highest degree due to its capabilities to create more complex, interactive and fluid applications. (Sun 2019, pp. 141)

In application and data layer development, e.g. backend-development, various options are available. Several commonly used programming languages suit well to backend development, such as Python, PHP, Java and JavaScript to name a few. (Web Developer Roadmap 2019) Backend frameworks, such as Django, Laravel, Spring and Node.js respectively, form an ecosystem on a language basis that enables rapid and efficient backend development. (Kaluža et al. 2019) The data layer can utilize relational, such as PostgreSQL and MySQL, or non-relational database systems, such as MongoDB and Cassandra, or both in tandem. (Web Developer Roadmap 2019)

In addition, based on the application requirements and architecture, the technology stack can be enhanced with various other tools. For example, data caching solutions exist, such as Memcached or Redis. In data caching pre-fetched datasets with high relevance are stored in the application layer to improve server response times (Mertz et al. 2018). To run the web application on the server, a web server, such as Apache or Nginx, is also required. (Web Developer Roadmap 2019) It is common to utilize containerization and cloud-based virtual environments to host the applications. With such means, the deployment process, configuration management and scalability of the application are enhanced. Cloud computing services include actors such as Google App Engine, Microsoft Azure and Amazon AWS. (Albrecht et al. 2017)

A combination of these various programming languages and frameworks is referred to as technology stacks. The selection of framework or programming language for each of the web application layers is profound as the decisions dictate the development ecosystem and suitable tools. The diverse technology stack of Airbnb is displayed in Figure 17.

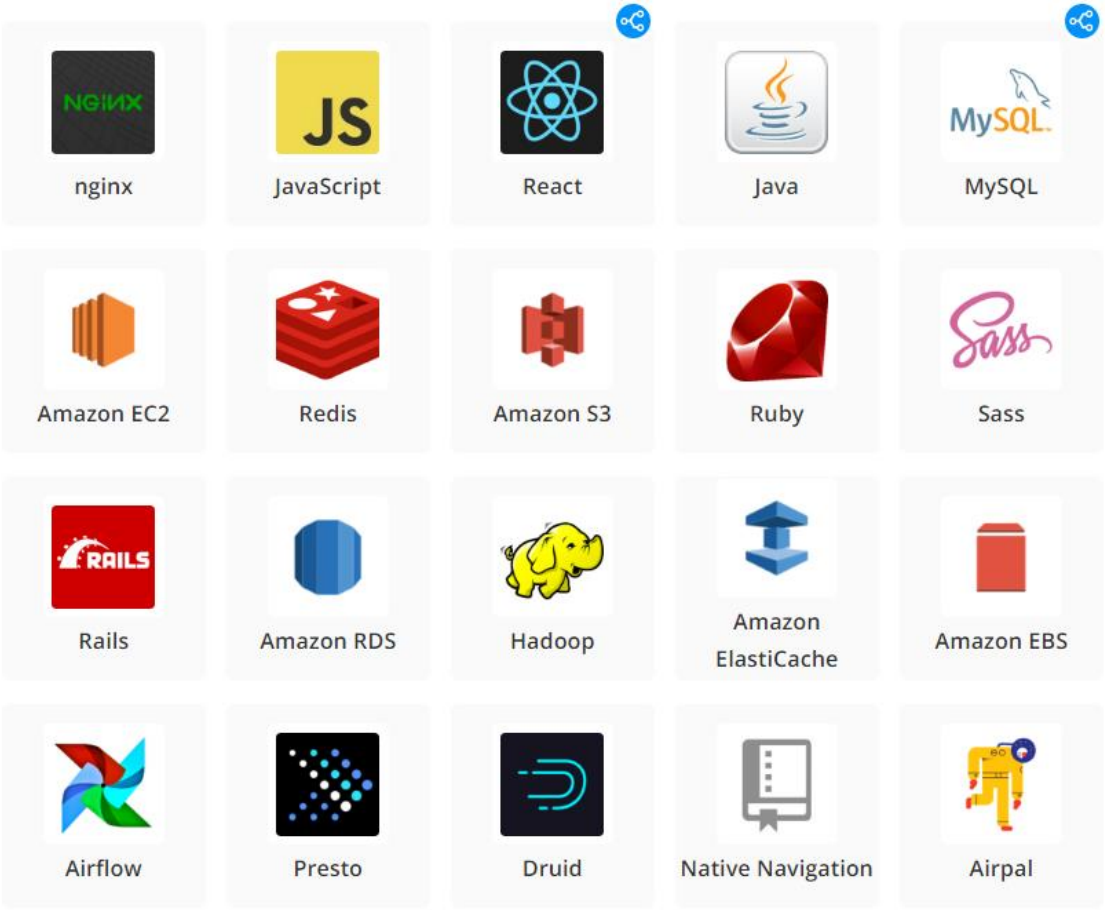

Figure 17 Technology stack of Airbnb (StackShare 2019)

3.3 Web application testing

Back in the day, testing of web applications was often retroactive, i.e. testing was initiated after issues or limitations were confronted (Murugesan 2008). Currently, web applications are targeted by even stricter quality requirements as the applications are more complex and handling critical aspects of our day-to-day lives (Fasolino et al. 2013). Consequently, the broad employment of web applications across various domains and the tendency of having short release cycles further amplify the need to carefully consider especially the structure of quality assurance practices (Leotta et al. 2013).

Testing web applications is complicated as there are various browsers and operating environments involved (Arora et al. 2012). Currently, there are various noteworthy actors in the browser market as displayed in Figure 18. The current market leader is Chrome with a market share of over 50 %. Numerous browser possibilities introduce complexities to testing as not all browser engines render content similarly or support all functionalities. This is emphasized as some applications might require support for old, i.e. legacy, versions of the browsers. In addition, websites are used more and more with mobile devices which requires responsiveness from the website content. (W3Counter 2019) The compatibility of the system for the different device and browser variants further emphasizes the need for usability testing (Fasolino et al. 2013).

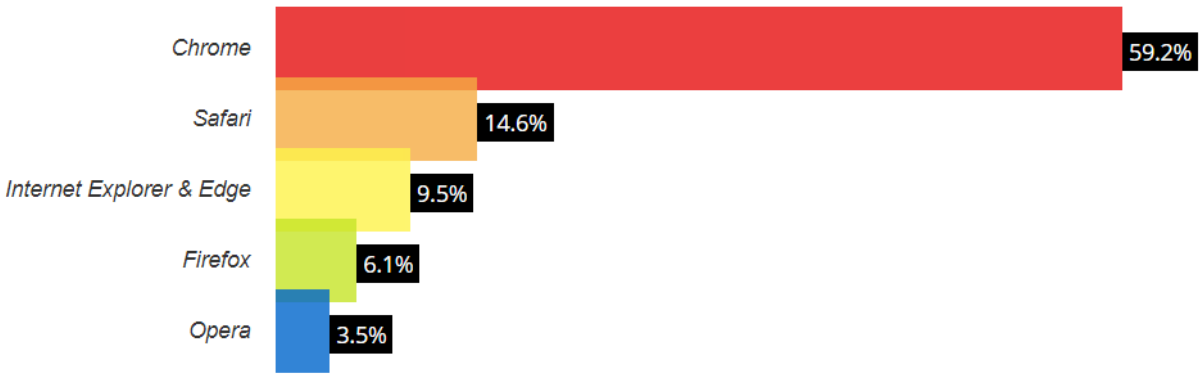

Figure 18 Browser market shares (W3Counter 2019)

In today's web applications views are rendered dynamically at runtime based on various inputs from the server-side and client-side logic and therefore identifying the correct layer(s) in which the defects are generated, is challenging. (Arora et al. 2012) This is emphasized due to the vast incorporation of third-party libraries in the application code as there is no guarantee of proper testing coverage and correct operation. Consequently, the inclusion of third-party libraries could introduce faults into the system. (Alenezi et al. 2016) As stated by Kappel (2006, pp. 133), the quality of web application is defined by the quality of the individual components and their interactions. This emphasizes the fact that various testing activities and levels should be practised to validate and improve application quality. The focus shall be especially put on regression testing. Due to the expectation of rapidly evolving

requirements, feature changes might be abundant. Regression tests should be in place to verify that the interaction with other parts of the system remains correct and subsequently no new faults are introduced after changing features or implementing new ones.

3.3.1 Testing levels

To support identifying the defect's layer and validating the quality of the application, each layer should be tested individually as well as in cohesion. According to Torchiano et al. (2011) research, the presentation layer of a web application is the most defect prone as about 50 % of web application defects trace to the presentational layer. This is speculated to be due to the complexity of the presentation logic, immature testing tools and the special execution environment of the web browser. As to discuss the test automation effort in a web application context, research by Contan et al. (2018) showed that the test automation division between unit, integration and UI-tests vary between web application projects and do not necessarily follow test pyramid model (Figure 9). Although generally, the larger emphasis was put on unit testing, it was observed that automation of functional tests through the UI was avoided due to fragility of execution and low return of investment. Therefore, it is necessary to consider test automation design on a product basis instead of following a model religiously.

In the web application context, the logic is sprinkled to various layers. Therefore, consideration of unit and component testing for all layers is beneficial. On the database layer, unit testing tools are scarce. However, validating the correctness of schemas, queries and procedures is key (tSQLt 2019). In the application layer resides most of the business logic, therefore unit testing is important. In the presentation layer, unit and component testing means range from verifying simple function implementation to comparing snapshots of rendered DOM (Document Object Model) elements of a UI component in the browser, i.e. snapshot testing (Jest 2019). The latter and more are made possible by modern SPA frameworks and their tooling which enable conducting a wide range of user interface testing via unit and component testing practices. Currently, in web application development the

language's standard or third-party libraries usually offer means to write and execute unit and component tests efficiently. (Vocke 2018)

Integration testing refers to the activity of validating the operation of units in their interactions. For example, this could include the activity of testing parts of application layer logic against a test database or interface of third-party service. (Vocke 2018) In web applications specifically, consideration of integration testing strategy and necessity is paramount as the layer and component-based structure complicates data flow (Di Lucca, et al. 2006). In addition, the introduction of actual dependencies, such as database, into the testing routine slows down and complicates testing (Duskis 2019). According to Vocke (2018), there could be a possibility that integration testing in web applications focuses on wrong parts, i.e. testing the used frameworks instead of application code.

According to Vocke (2018) automated end-to-end testing of web applications is difficult, especially through UI. Browser quirks, timing issues, animations and unexpected popups complicate the testing process and a lot of time is spent on debugging the tests. Slow execution time and high maintenance cost further steer testers to automate only the testing of user paths that are considered most valuable. A wide variety of open-source and commercial testing tools exists, that enables browser-based end-to-end testing automation. According to Leotta et al. (2013), automated end-to-end tests are either programmable- or capture-replay tests. In programmable tests, the test is programmed manually and in capture-replay testing the tests generated automatically from a recording of user actions. Consequently, the implementation time for programmable tests is higher but the maintenance is easier and therefore with subsequent releases programmable tests triumph in cost-effectiveness. These maintenance activities consist of responding to presentational or logical changes of the user interface.

Manual end-to-end testing enables detecting system issues in a straight-forward, albeit in a repetitive and possibly cost-inefficient manner, especially when considering complex web applications. However, the detection of usability issues and smells is difficult without testing the system end-to-end manually. (Grigera et al. 2017) Generally, usability testing begins

from the wireframe- and prototyping phases and extends to end-to-end testing. Depending on the system and project context, alpha and beta testing, as well as user-acceptance testing, could be viable options to further gain feedback and support end-to-end testing activities by utilizing the end-users. (Crispin et al. 2009)

3.3.2 Performance-, load- and security testing

Performance of web application concerns all layers, database, application, and presentational layers. According to Parzych (2016) and Loadster (2019), 80 % of waiting time is front-end based and 20 % back-end based. However, such generalizations are context-dependent and during peak loads, wait time ratio swifts more towards backend (Loadster 2019). Additionally, the server and network infrastructure affect web application performance. Structure of tables, the efficiency of database queries and complexity and efficiency of the application logic as well as the server architecture define the pace and efficiency at which the backend can handle requests originated from the users. (Parzych 2016) Furthermore, the size of the front-end application bundle, e.g. template, script and media files, defines how quickly users can interact with the system initially. In addition, general browser rendering performance can be majorly affected by inefficient code and memory leaks. (Front-End Checklist 2019) Mediocre performance imposes the possibility of users not reaching the service, having long wait times or interacting with an unresponsive system.

By the means of performance testing, such issues can be identified, and performance requirements validated. According to Matam et al. (2017) load testing allows inspecting the performance of web applications. Load and stress testing focus on inspecting system behaviour with different loads, such as expected or peak load. Consequently, load testing is a mean to identify the system's maximum capacity, referred to as capacity testing. Web application load testing tooling enables the execution of massive amounts of HTTP-requests towards the application server and logging performance reports of them (Tikhanski 2018). Additionally, front-end performance can be inspected with developer tools built-in to browser or external software.

A wide variety of security issues haunt web applications. OWASP (Open Web Application Security Project, 2015) is a community that provides information related to web security. Currently, most fundamental security issues concerning web applications are:

- Injection
- Broken authentication
- Sensitive data exposure
- XML external entities (XXE)
- Broken access control
- Security misconfiguration
- Cross-site scripting (XSS)
- Using component with known vulnerabilities
- Insufficient logging & monitoring

These issues might arise from faulty design or implementation. These faults could be created in-house due to neglecting security aspects or by exercising faulty third-party code. Additionally, a faulty server configuration or software execution environment could introduce vulnerabilities. (OWASP 2017) In security testing, the focus should not be in penetration testing activities, e.g. trying to exploit the system and detect vulnerabilities. Instead, possible security issues should be mitigated by taking security aspects into account during design, development, deployment and maintenance stages. (OWASP 2015, pp. 24) A wide variety of open-source and commercial tools are available that enable identification of possible security issues. These tools support manual security testing, enable automatic issue detection by static code analysis and inspecting executing software. (OWASP 2015, pp. 214-216)

4 EMPIRICAL RESEARCH

In the following chapter, the construction, execution and results of the empirical research are outlined. The qualitative multi-method research, consisting of initial literature review, questionnaire and interviews, was conducted in April and May of 2020. Due to the COVID-19 pandemic, the interviews were conducted remotely.

4.1 Research methods and background

The literature review of software development and testing in agile and web application context is the theoretical background for the development of an empirical survey. The empirical survey of the thesis consists of a questionnaire and interviews. According to Meyer et al. (2001, pp. 4-5) expert judgement is a common method to model and solve technical problems. It is especially suitable for interpreting decision-making processes and modelling the current state of an issue (Meyer et al. 2001, pp. 4-5). Both of which are of high interest in the context of the thesis. In an agile context, the level of cooperation is high, and testing is a joint effort. Therefore, the evaluation needs to account for various viewpoints. Developer and testing oriented project members, as well as project managers overseeing the operation, all have experiences on benefits and possible issues of testing from different viewpoints.

The research findings of the thesis are qualitative. By researching with a multi-method approach, triangulation is achieved. Triangulation is a way to increase the creditability of qualitative research by utilizing various research methods (Cohen et al. 2007). The research methods of the thesis, literature review, questionnaire and interviews, all contribute to the understanding of the researched phenomenon. By combining various research methods and consequently utilizing multiple sources of information, comprehensive answers to the research questions and manifold discussion can be created.

The research process resembled waterfall, as the stages were completed consecutively. Firstly, the phenomenon was researched from literature. Secondly, the most significant

project factors affecting testing decisions were extracted via questionnaire. Thirdly, the literature review and the questionnaire results were utilized in the development of the semi-structured interviews. After conducting the interviews, the collected data was analyzed and conclusions, as well as discussion, were derived.

The data for the survey is acquired in collaboration with Visma Consulting Oy, specifically in the context of its Product Creation Services (PCS) unit. Titled OCTO3 Oy before the business acquisition by Visma Consulting Oy in 2017, the unit focuses on designing and developing custom software solutions that cater to the various needs of both public and private sector clients. Majority of the unit's workforce of approximately 80 employees consists of developer-oriented software engineers with varying specializations. These developers are accompanied by sales and project managers as well as user experience and service design professionals. Most of the software designed and developed by the unit is for web or mobile platforms. Categorization of the projects that are executed at PCS is difficult since the projects range from short-lived and small projects to multi-year projects with a high number of involved personnel. Also, the company offers subcontracting. Knowledge transfer has been emphasized in the unit's strategy, as in fast-paced project work, it would be beneficial to utilize the previously identified best practices and solutions in subsequent projects. The research theme originated from such need and the target of the research is to model the testing practices that are deemed worthwhile and to extract the information on how the practices should scale within different project contexts.

4.1.1 Questionnaire

A questionnaire is a commonly used instrument in survey research. The questionnaire instrument for the research was developed in conjunction with the best practices of questionnaire development. (Trobia 2008, pp. 652-655) Additionally, the target was to create as minimalistic questionnaire as possible to enable higher response rate. Questionnaires are straight-forward to conduct and analyze and they reach a large audience with small effort. These were the primary drivers why the factors were extracted via questionnaire. Also, the

questionnaire enabled every interested individual from the collaborating company to take part in the research.

The conducted questionnaire is in Appendix 1. The questionnaire format consisted of introduction, instructions, and questions. Three demographic questions were included. Current role in the organization, as well as the experience in software industry and web application development, of the respondent, were gathered. Only one primary question was included. The primary question was open-ended and required to name five to ten most significant factors that affect the testing decisions in web application projects.

The primary question of the questionnaire was left open-ended due to the complexity of the topic. Close-ended evaluation of predefined factors was considered but later deemed unfeasible since the questionnaire could have become repetitive and taunting and some significant factors could have been left out of consideration. However, the general categories for the factors were on display to arouse consideration. These categories originated from Clarke et al. (2012) comprehensive literature review in which they identified 44 situational factors that affect the software development process (Figure 19). These factors are grouped into eight different categories as depicted in Table 1. Also, these factors are further divided into sub-factors, totalling at 170 different factors.

Table 1 - Situational factors affecting the software process (Clarke et al. 2012)

Category	**Description**
Application	Characteristics of the application(s) under development
Business	Strategic and tactical business consideration
Management	Constitution and characteristics of the software development management team
Organisation	Profile of the organisation
Operation	Operational considerations and constraints
Personnel	Constitution and characteristics of the non-managerial personnel involved in the software development efforts
Requirements	Characteristics of the requirements
Technology	Profile of the technology being used for the software development effort

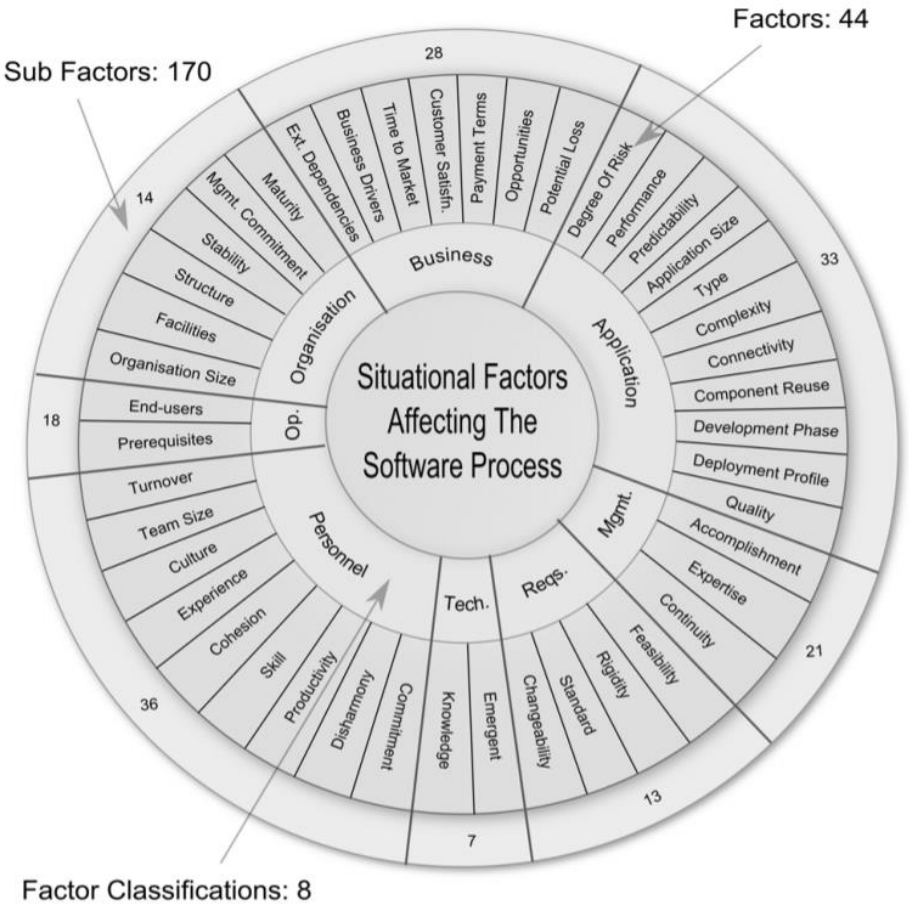

Figure 19 Situational factors affecting the software process (O'Connor et al. 2016)

The questionnaire was created with Google Forms. This was due to the company using Google's organizational accounts. The questionnaire was anonymous but login with organizational account was required to avoid duplicate responses and responses outside the target group. Before distribution, the questionnaire was pilot tested with one individual from the target group to verify its soundness. The invitation for the questionnaire was put out via email and subsequent reminders were shared in instant messaging application Slack. The target group consisted of 81 individuals. The questionnaire was open for one week.

The questionnaire received 17 answers in the one week: 11 from developers, 4 from architects/specialists and 2 from superiors. All respondents had worked in web application projects. 10 respondents answered with own words, two combined the predefined categories

with own words and four selected from the predefined categories. The analysis focused on the responses with own words.

The answers were coded into categories, that naturally seemed fit, and the occurrences of similar answers were quantified. The categories originated from the data, but the work of Clarke et al. (2012) undeniably influenced the answers as well as the categorization. With such a limited amount of responses, quantitative analysis of the data was not the primary interest. The significance of the factors was based on the number of similar factors by different respondents. As such, the number of similar factors was of interest. Multiple occurrences of similar factors further validate the significance of the factor.

Overall, the categorization of the answers was a challenge due to the diverse nature of the answers. All the respondents' roles were represented in all the categories, which indicates that no significant differences were due to the role demographic. The demographic information of respondents' experience indicated that most of the less experienced respondents did not opt for answers with own words. Due to the limited number of responses and the consequent risk of identifying individuals, the demographic information was not utilized in the analysis.

4.1.2 Semi-structured interviews

The purpose of the semi-structured interviews, in the context of the thesis, is to extract information on how web-applications should be tested and how different project contexts influence the way the testing should be orchestrated. Semi-structured interviews enable extracting various viewpoints in a short time period. Additionally, as the interviewed individuals stem from various backgrounds and experience levels, the semi-structured nature of the interviews enables all to answer with their best ability and there is room for open discussion. The anonymity of the interviews also encourages to discuss matters unfiltered.

Nine individuals from the collaborating company were interviewed for the research. The interview group consisted of specialists, developers, and project managers. The specialists

have a strong background in software development and along with full-stack development activities, they consult the decision-making processes in the company. The professional software engineering working experience of the interviewed personnel averaged at 11 years. Web application project experience averaged at 7 years. The interview duration ranged from 40 minutes to 80 minutes, averaging at 66 minutes. The interview material, recordings, and notes were destroyed after the analysis was finalized. The interview structure is included in Appendix 2.

The themes for the interviews were derived from the literature review. The Agile Testing Quadrant by Crispin et al. (2009) is the fundamental theoretical origin for the interview structure. Additionally, the testing level models, as well as the overall literature on web application testing, influenced the themes. The interview themes were the following:

- the overall significance of testing,
- testing levels (unit / integration / system),
- manual and automated system testing
- non-functional testing
- supportive practices for testing (CI/CD and TDD)

Also, the interviews are focused on how such themes are affected by different project contexts. The different project contexts are defined by the various factors extracted with the questionnaire. As the timespan and resources for the research were limited, the research focused only some of the most significant project factors, which were:

- Budget
- Criticality
- Schedule
- Personnel know-how
- Technology (availability and limitations)

4.2 Questionnaire on project factors that affect testing decisions

The questionnaire resulted in a wide range of significant project factors that affect testing decisions in the web application context. As a result of the questionnaire analysis, five factor categories were identified:

- Requirements
- Technology
- Development team
- Management
- Customer

Furthermore, the factors that occurred in two or more responses are grouped into Figure 20. Some of the factors were not distinguishable for specific categories and therefore interpretation of such factors concerning the most fitting categories was derived. Figure 21 highlights the recurrence of budget in the responses and the overall widespreadness of the responses.

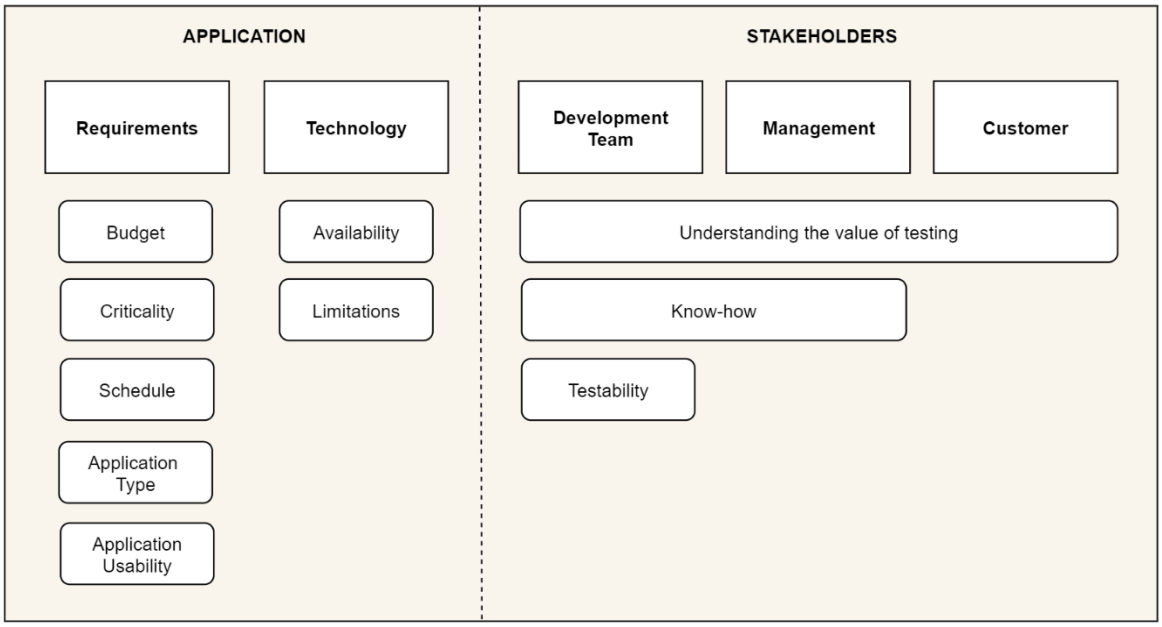

Figure 20 Recurring factors affecting testing decisions from the questionnaire

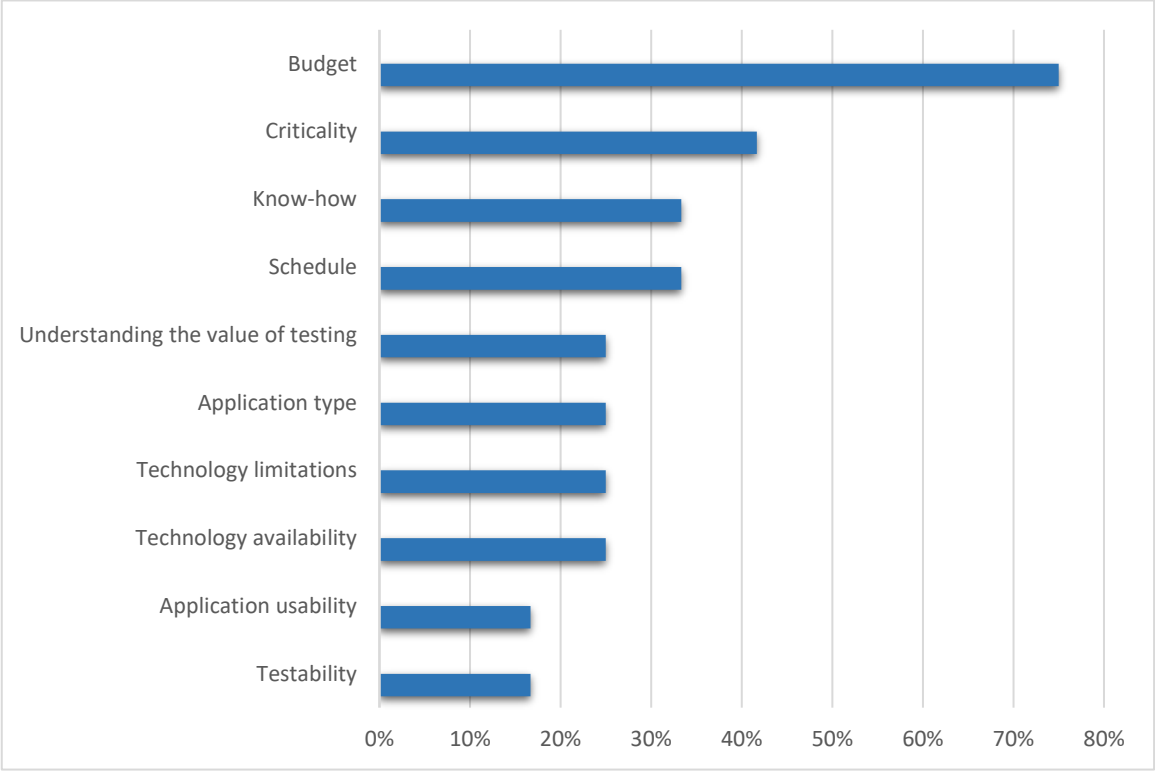

Figure 21 Percentage of questionnaire participants responses containing the recurring factors

The requirements category is dominated by factors budget, criticality and schedule. The project budget was included in almost all the respondents' responses. The budget and schedule were commonly combined into one response. However, one response also identified the length of the project as a separate factor. Some also specified that the budget and schedule pressures and the lack of budget influence testing decisions.

The criticality of the system turned out to be one of the most significant factors. In addition to the general criticality factor, other dimensions were also included, such as criticality of the application and criticality of the application for the customer. The fourth factor in the requirements and application-related category is the application type. Themes such as the application being UI or server-oriented and client or business targeted emerged with this factor. Also, the application usability needs seem to play a role in the testing decisions. Further, some of the singular responses include application security, estimated lifespan and the rate of requirement changes.

The technology category consists of the availability of the testing frameworks and tools as well as the limitations that the development technology for the project could introduce. The know-how aspect is also present, as a lack of knowledge on testing tools and practices is viewed as a significant factor. The development team factor category is know-how centric. Multiple answers shared the idea of the experience and knowledge of the individuals influencing testing decisions. Accompanying the general answer on the factor, some further specified that the developer know-how on testing is significant. Some of the answers also went into technical details of the development. As such, the testability of the system emerged as a factor. This was deemed as the responsibility of the developers.

The management category consists of a wide variety of factors. Prioritization, development process and preparedness to maintain tests and their operation are examples of factors that are distinguishable for management. Themes on testing orchestration emerged as well. Shared responsibility and early planning of testing design, implementation and material were deemed significant. The recognition of testing the most significant parts of the application is also a factor. In addition, the number of personnel involved in the project influence testing

decisions. According to the questionnaire, customer-related factors influence testing decisions. In addition to the general customer answer, a couple of more insightful responses were given. The willingness of the customer as well as the level of understanding of software development plays a part in testing decisions.

Additionally, some factors matched to multiple categories. One of which is the understanding of the benefits of testing, that involves all stakeholders. One of these answers highlighted the factor with a negative connotation, that there is no understanding of the long-term benefits and value of testing. Furthermore, the know-how of the individuals working on the project is of interest. Such factors concerned the experience, knowledge and education level of the management and development team. Lastly, singular responses such as familiar practices and appropriateness of testing popped up.

4.3 Interviews on testing practices in web application projects

The following section summarizes the interviews on the testing themes that were derived from the literature review. Discussions on significance of testing, testing coverage in different levels, automated and manual system testing, non-functional testing, and supportive practices are summarized.

4.3.1 Significance of testing

Overall, the significance of testing web applications in the research group is considered to be high. The significance of testing web applications stems from various aspects. Fundamentally, testing enables to validate that the software is sound against the requirements and its quality meets the needs of the project. The significance of testing is highlighted by the fact that it enables to detect regression. Specifically, in modern agile web application projects, the features are usually developed swiftly and built upon previous ones, which introduces the risk of breaking existing features. As such, testing is viewed as a method to confidently introduce changes to the system which inherently is fundamental for agile software development. In addition, a comprehensive set of well-written tests act as a

documentation of the system. Testing also enables the product to advance technically and from the business perspective.

The significance varies in different project contexts, but the benefits of systematic testing are deemed to be useful in any kind of web application project. This is pronounced by the fact that the scope of the web applications could creep. Initially small project scope could escalate in the future and without foundationally solid testing practices, the project progress could be deteriorated.

The complexity and criticality of the system were considered to further signify the need for testing. In addition, web applications are used in various browser and operation environments which highlights the testing need. The non-functional aspects of web applications are also underlined nowadays which requires validation of them with testing. Some referred to the fact that the testing significance depends on how tolerant the project is on bugs and issues that emerge in deployment. One interviewee also discussed that the contractual liability for defects should be considered in the testing decisions.

As to how much project effort should be allocated for testing, the answers varied from 10-50 %. The generally optimal effort was viewed to be at 20 %. Such a testing investment was deemed worthy due to these various benefits of testing. Neglecting the testing during the project execution had been proven costly on many occasions, which further justifies the testing investment. However, some argued that too extensive testing could limit the pace at which the features are developed, i.e. the velocity of the development. It was highlighted that customers, in many cases, expect new features and visible changes instead of quality improvements and validation. As such, from the customer perspective, features are the key deliverables and the main target of the software development process. Such a theme pushes the testing into the sidelines. However, nowadays the customers are becoming more and more educated about web application development which means they expect better quality products as a default.

One of the key concepts in testing significance was deemed to be the accuracy of the system and feature specification. Fundamentally, testing is reliant on the requirements and their acceptance criteria. Without an accurate and specific description of how the system or feature should work, the testing is inherently difficult.

4.3.2 Testing coverage in different levels

Overall, none of the testing levels was deemed less useful than others. Fundamentally, to achieve high quality – all testing levels should be covered. Unit and integration testing were referred as the more approachable and realistic method, especially in the unit's context, as such tests can be implemented by the developers and are easier to integrate to the development process. The unit and integration tests are also, in most cases, cheaper to implement and maintain. Generally, the lower level testing was viewed as a beneficial method to validate the edge cases and as such remove burden from the testing at the higher levels.

Interestingly, system and E2E testing are viewed to bring great value to the testing, especially when automated. System testing level was highlighted since modern web applications are usually complex single-page applications and are composed of several components, services and often require many integrations. The main benefit of system-level testing was discussed to be the ability to detect regression extremely efficiently. It also essentially the easiest way to test the functionalities of the system.

Additionally, some distinction was made between the backend and frontend of the web application concerning the testing levels. Generally, the lower level testing in the backend was viewed as a straight-forward and fundamental practice. However, in the modern and complex SPA frontend, system-level testing usually reveals issues efficiently. Based on the research participants experiences, writing too complex low-level tests for the frontend flow should be generally avoided as they yield less value when compared to the implementation cost. There is also a distinction on whether the web application is built from a scratch or it is a legacy product. Writing low-level tests to existing software afterwards is difficult and

therefore for legacy applications the system testing level might be the only viable option. With new products, the inclusion of lower-level tests is inherently easier as code is written from scratch.

4.3.3 Automated and manual system testing

Optimally, most of the system testing effort should be automated and complemented with manual testing. High emphasis was put on automated system testing as it minimizes the manual labour and issues that arise from humane factors. It was deemed to be the go-to solution to detect regression efficiently. A great number of different use cases and paths in the applications can be tested continuously during the development. Automation of the functional tests cases enables the test effort to focus on other aspects, such as necessary non-functional aspects.

However, the automated system testing was felt to be underutilized in modern web application development. Many reasons contribute to this. Firstly, the initial investment in setting up the infrastructure of automated system testing is high. Secondly, the test implementation is usually labour-intensive. Thirdly, automated system testing requires specific know-how from the project individuals. And finally, the benefits of automated system testing usually liquidate only in the long haul. Such trade-offs usually hinder the utilization of automated system testing and initially, the manual counterpart feels tempting. However, various research participants discussed that many projects could have benefitted from automated system testing in hindsight.

Also, it was highlighted that great consideration is required as to when implementing the automated tests as in agile context, the features are often ever-changing. The implementation of an automated system test requires high time investment and as such the feature should be stable before the implementation of such tests. With initially detailed requirements or fast stabilization of the changing features, the automation process of the system testing is streamlined. The less maintenance the automated system tests require the more cost-effective they are.

Manual system testing was discussed to be efficient in testing highly complex scenarios that are hard to automate or in cases where human execution reaps benefits. Examples of such cases include complex user interfaces and usability considerations. Optimally, the primary function of manual system testing was to be utilized as a complementary method for automated system testing with smaller emphasis. However, in various web application projects, most of the testing effort is still manual. In manual system testing, most of the issues are humane. Human is rarely consistent in repetitive manual labour which leads to issues as the tests might not be executed systematically every time. Further, especially if the feature is tested by its developer, there could be bias, lack of expertise or interest which affects the testing process.

4.3.4 Non-functional testing

Generally, the interviewees agreed that testing of non-functional aspects of a modern-day web application cannot be neglected. Some discussed that the customers are becoming more and more educated on such aspects and therefore the specific requirements towards non-functional aspects of the application are not out of the ordinary. The research participants highlighted especially security and performance attributes of the applications. Some participants referred to the fact that in the company context, service and user experience design are strongly advocated and integrated into the software development process and therefore the usability aspects usually require less consideration. Additionally, some discussed that most of the usability issues are usually identified during the development process – especially if the team knows the application's business domain. The usability concerns are usually emphasized if the web application is targeted for the masses.

The security aspects of modern-day web application were considered critical. Usually, in the simplest of an application, there is authentication and possibly various access levels and user rights. As such the access to the system's data is limited to specific individuals. Breaches in the application logic could jeopardize the system's data. Depending on the data criticality, such aspects could lead to catastrophic consequences. Also, it was highlighted, that the seemingly external factors should not be underestimated. For example, the operational

setting, such as how securely and where the system's server and databases reside, all contribute to the security of the system. Additionally, the research participants discussed that the usage of third-party libraries could introduce security issues to the system and as such great care and constant monitoring should be invested into the selection and utilization process of such libraries.

As to discuss the performance aspects, some referred to the fact that modern web application technologies are performant and therefore many applications could handle a lot more traffic with ease. Additionally, the validation of base-level performance testing of web applications was considered to be quite straight-forward in multiple cases. In performance testing, fundamentally it is advisable to mimic the expected production environments and datasets during the development to avoid issues in the production. The expected number of concurrent users and the expected sizes of the databases were discussed to be key indicators of how beneficial performance testing is.

The challenges of non-functional testing are manifold. In most cases, the functional aspects of web applications are usually the primary test targets. This usually leaves the non-functional aspects to the sidelines. The research participants advocated that efficient testing of the non-functional attributes of the system require special outlook and expertise on software development. However, in web application projects with educated project members, the base level testing of such attributes is usually covered during the functional testing. It was discussed that many of the non-functional issues can be identified during manual testing with an educated pair of eyes. With special tooling and automation, non-functional testing can be brought to a more advanced level. Automation of non-functional testing is difficult since every attribute requires different tools and associated expertise. Therefore, a more advanced non-functional testing setup is in most cases costly and therefore rarely utilized.

As a summary, the non-functional aspects of the web application boil down to one having a sound design and constant monitoring. Most of the issues can be avoided when the non-functional aspects are considered during the planning and design of the web application.

Consequently, monitoring of the non-functional aspects of the software should be integrated into the software development process to identify the possible issues early. In a more advanced non-functional testing setup, it is desirable to make the numbers, such as performance metrics, visible for the project team to further monitor the state of the application.

4.3.5 Supportive practices

Continuous integration and deployment were referred to be a highly significant supportive practice for testing. The research participants agreed that setting up a continuous integration pipeline should be almost an automaton. It was discussed that only in the small, 'deploy once' type of projects, the cost of setting up the CI-pipeline could be initially too high. The more complex the build and test process, the more beneficial it is. In addition, it was discussed that the bigger the development team and the more releases are expected, the more critical CI is. The benefits of continuous integration are abundant. Firstly, setting up continuous integration saves time. Manual deployments are usually a hassle and with automation, the humane errors during the deployment can be avoided. Secondly, the continuous integration pipeline motivates to set up different environments for development, testing and deployment and generally to deploy smaller changes at a time. As such, the system configurations and environments are managed during the development and the manual testing is facilitated with ease. CI-pipeline also acts as self-explanatory documentation of the build and deployment process. For example, in cases where project member composition changes or a long-standing project in a maintenance/operational phase requires further development, such setup is helpful. Finally, the continuous integration pipeline facilitates the continuous execution of the automated tests.

The research participants had rarely utilized the practice of test-driven development. Some discussed that the timing of writing low-level tests is highly dependent on the developer's practices and whether the requirements are stable. Fundamentally, the key point is to write the tests – regardless of whether it's before or after the feature implementation. However, most of the participants discussed that TDD is essentially a sound practice but requires

expertise and a certain type of project setting. Essentially, the practice leads to high testing coverage. However, the requirements and design of the system/feature must be highly accurate to enable the development team to write and design the tests beforehand. As such, it was discussed the more realistic use cases for the utilization of TDD in the web application context relate to standardized components and interfaces. Generally, the backend logic is usually more fitting for the practices of test-driven development. Additionally, some advocated that the utilization of practice such as test-driven development requires the collaboration and investment of the whole development team which further highlights the need for skilled project members.

Various other practices were discussed that support testing. Firstly, even a small testing plan was discussed to be the fundamental foundation to enable systematic testing. The testing plan should include the testing focus areas and responsibilities and how testing is integrated into the sprints and development. The testing plan should be revised and monitored during the project execution. In addition, the project budget and schedule should account testing right from start. Especially in software consulting context, the sales deal formulation is essential in enabling testing to be considered in the project scope. Also, at the feature level, the task estimations should include the testing effort and the type of testing.

Secondly, the management of environments is critical. Inclusion of separate testing environment(s) essentially streamlines the development and testing process. Especially, if the application deployments are customer-specific, the significance of proper environment management increases. Thirdly, the utilization of peer reviews is a straight-forward and an efficient way to further validate the application. Finally, the monitoring of testing could enhance the process. However, in general, the research participants downplayed the traditional testing coverage metrics and advocated to focus on monitoring whether the development team is testing the right things with the right means.

4.4 Interviews on project factors impact on testing

The following section summarizes the interviews on the five most recurring factors that were extracted from the questionnaire, namely budget, criticality, schedule, know-how and technology.

4.4.1 Budget

The project budget fundamentally defines to which extent the application is tested. Some referred that in an optimal situation the impact of the budget for the testing effort would be minimal. However, the participants felt that in many cases the project budget is limited to the delivery of the features and as such the testing effort remains minimal.

As to discuss which testing practices are viable with a limited budget, the research participants mandated that the testing focus should be especially on the critical application features. In such a situation, focusing on manual system testing is a viable option as a compromise. This setup could be paired with a couple of unit or integration level tests. The testing focus would be on the happy paths. Although such a testing setup was deemed insufficient if the application is not trivial. With a higher project budget, the testing effort would swift towards automation and more coverage could be delivered. In addition to the happy paths, the edge cases and non-functional aspects could be tested to a higher degree. As a result, a higher quality product could be delivered.

4.4.2 Criticality

Application criticality emphasizes testing. The more critical the application, the more testing effort is required. The most common examples of critical applications are applications that affect the users' health, finances or handles critical business functions. The more catastrophic are the consequences of a faulty system, the more critical the application. However, as an opposing viewpoint, it was discussed that all applications are critical as the customers are paying for them. The criticality of the application emphasizes the testing effort

in general - to deliver a quality product for the customer and to avoid reputation damage. Yet, as various real worlds examples prove, the criticality of the application is not a guarantee that the testing investment is, as it should be, sufficient.

4.4.3 Schedule

The research participants felt that the schedule and budget are tightly coupled as time is money. With a tight schedule, the testing is often overlooked due to the features being more critical. However, in some cases, it is justifiable from the business perspective, as quick release to the market could be the key to success regardless of the quality. The discussions revealed that tight schedule could pressurize to overlook the non-functional aspects of the web application as the focus usually shifts to them in the latter stages of the project. Overall, tight schedule was viewed as a negative factor for the application quality as it motivates to cut corners in development and testing. For a project with a tight schedule, the testing practices remain similar as with a small budget – the focus is on happy-path testing.

4.4.4 Know-how

The research participants revealed that testing of web applications is highly know-how dependent with various dimensions. First of all, testing requires a specific kind of mindset and attitude, especially from the developers, as they are critiquing their creations. The software development and testing know-how directly transfer into a better-quality product. Knowing what to test, when to test and how to test generally stem from the individual's know-how. The field of testing is filled with a wide variety of different test types and tools that are constantly evolving with development technologies. Dangling in such an environment requires expertise. Some also referred to the fact that testing is attitude dependent, as some developers do not necessarily value the testing activities.

Efficient writing of lower-level tests, such as unit and integration tests, requires that the code is testable. Some discussed that inexperienced developers design and write code in such a way that introduces unnecessary complexities and dependencies. Without refactoring,

efficient writing of the lower level tests for such a codebase could be difficult. Similarly, in the system testing level, the similar know-how related issues arise. The efficiency of manual system testing is highly related who is doing the testing – how experienced the tester is to identify issues from web applications. In addition, automation of system testing requires specific knowledge on how to utilize the tools. Similarly, the application markup must be accessible for efficient writing of automated E2E tests. Even more so, testing of the non-functional aspects requires a wide range of expertise on web applications in general and specific knowledge on the tooling.

In a hypothetical scenario, where the web application project is limited by know-how, the interviewees discussed that mentoring could be a viable option. Such a project would benefit from someone who shares know-how and as such kickstarts and discusses the testing in the project's context with the project members. Interestingly, in the projects where the project personnel are highly experienced, the significance of testing does not decrease. Quite the opposite - as the research participants referred to emphasizing of automated and non-functional testing.

4.4.5 Technology

The technological aspects of a web application project play a role in testing consideration. Some participants referred to the fact that the programming language lays the foundation on the testability of the web application. Statically typed languages generally encourage to maintain the testability of the software. Then, the selection of development technologies further influences the tooling which can be used and is available for the unit and integration testing. Overall, the participants felt that the development technologies and frameworks support testing and the tooling scene is diverse. However, the diversity of test tooling complicates the process of finding the right tools for the job. As a rule of thumb for the harmony of technology and testing, some referred that if the development technology hampers testing, the development technology is wrong.

4.5 Other themes that emerged during interviews

4.5.1 Testing culture

During the interviews of the empirical research, the effect of testing culture on testing considerations became evident. The research participants discussed that testing culture fundamentally boils down to whether the personnel in the organization understand and value the benefits of testing. With a decent testing culture, the testing is integrated into the organization and higher quality applications are developed. In addition, some addressed how to develop a good testing culture. Test planning and clear responsibilities are the fundamental origins of good testing culture. Enabling continuous learning and knowledge sharing of new tools, concepts and methodologies further enable the testing practices to develop. Some discussed that testing is a question of attitude – every software project should be approached with an attitude that we are building a good product. Especially for less-experienced software developers, it would be beneficial to learn the best practices right away regardless of the project context. Some developers also advocated the "developer experience", as the employment of mature testing practices further increases the meaningfulness of work.

5 DISCUSSION

The following chapter is reserved for discussion of the research results and the implications of the results. Additionally, future research directions are presented.

5.1 Defining testing practices for an agile web application project

The following section discusses the research questions by synthesizing the information gathered from empirical research. A summary of the project factors affecting testing decisions is depicted and definition of suitable testing practices is briefly discussed.

5.1.1 Project factors affecting testing decisions

The projects factors and subfactors that affect testing decisions are depicted in Figure 22. These factors were collected with a questionnaire and interviews. A wide variety of factors contribute to the overall test process that can be implemented for a web application project. During the empirical research, project budget, schedule, criticality, personnel know-how and complexity were highlighted. The factors originated from various categories which further amplifies that the testing of web applications is a complex problem and affected by various dimensions.

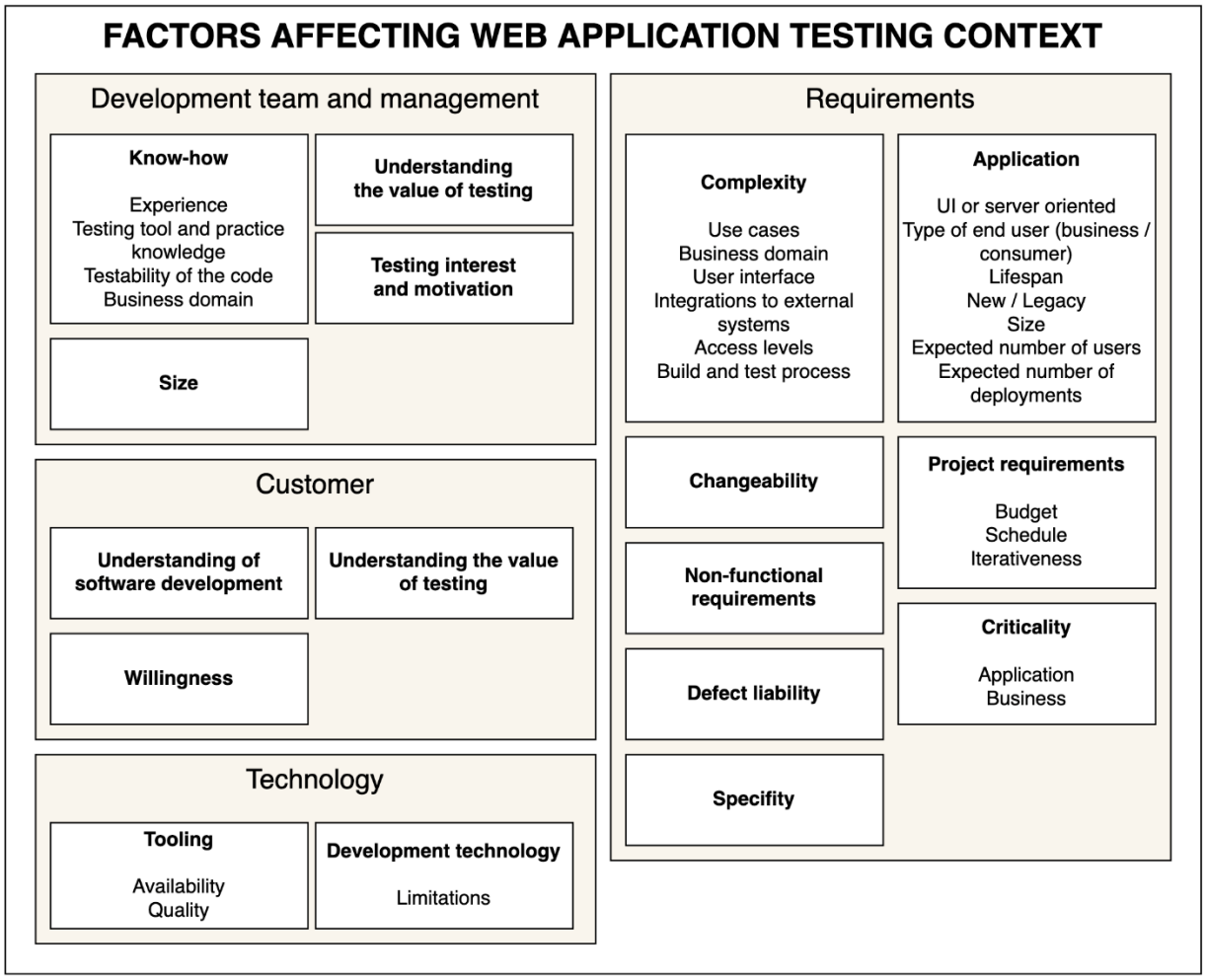

Figure 22 Factors affecting web application testing context

5.1.2 Defining suitable testing practices for an agile web application project

During the research, it became evident that test planning is the key to succeed in scaling the testing practices to a sufficient level. The high significance of testing further amplifies that testing is and should be part of the software development process. Generally, the research validates that the testing in agile web application context requires a wide range of practices and a great sense of collaboration. Regardless of the size of the testing investment and constraints, it is essential to choose suitable practices and implement them systematically. With a systematic testing setup, the functional and non-functional quality of the application can be validated, and regression can be identified and controlled.

As to discuss the testing levels, the system level testing is pronounced in the testing of web applications. Whether automated or manual, such testing enables efficient functional validation and regression detection. Also, it enables the detection of non-functional issues to a degree. However, none of the testing levels are to be excluded as a comprehensive test suite consists of tests on all testing levels. In addition, the non-functional aspects of the application require great consideration. Identification of the critical non-functional quality attributes for the specific application is essential.

In addition, the research discussed the various benefits of continuous integration and deployment pipelines, timely implementation of the tests and test automation. As such, the research encourages to utilize build, test and deployment pipelines and essentially the automatization of tests. Timely implementation of the tests is critical to avoid extra work. It is also essential to consider to which degree test automation is utilized.

5.2 Relation to the literature

Fundamentally, the empirical research agrees with literature, that testing definition is a complex problem. The literature review as the empirical research both fundamentally highlights similar factors as the primary drivers in the testing definition. Black (2009) discussed the linkage of four project elements (Figure 5), quality, features, schedule and budget. Fundamentally, the empirical research discussed the same phenomena, as the project budget and schedule were identified as the primary factors affecting testing decisions and it was discussed that the features are, in some cases, more important than the quality. Similarly, the research revealed that the testing context of web applications is affected by various dimensions, as the project factors identified from the empirical research originate essentially from all of the Clarke et al. (2012) software context factor categories.

Both the literature, previously conducted research and the empirical research of the thesis highlights the role of test planning. Testing is such a pervasive practice that it should be integrated tightly into the software development process right from the get-go. Similarly, the

research participants shared a similar vision on the optimal testing setup in web application projects as the literature. Such a testing setup consists of highly automated testing with only supplemental manual testing. In addition to functional testing, the non-functional aspects are also tested systematically. As to discuss the non-functional testing in relation to software product quality, the research focused on the security, performance and usability characteristics of the system. These characteristics were also highlighted by the research participants. However, the maintainability and portability characteristics also were discussed. Testability of the system was highly emphasized as an enabler for testing. Also, the discussion on build pipelines further emphasizes the installability aspect of the product.

The scope and methods of web application testing are wide which further validates that the Agile Testing Quadrant (Crispin et al. 2009) is still relevant. Especially in the web application context, as all the quadrants are relevant. The quadrants can be used as a helpful tool in the definition of suitable testing practices. The testing maturity levels in the web application context are fascinating and multi-dimensional. The advancements in testing maturity are not unambiguous. According to the research, to achieve more advanced maturity levels, the project personnel must be highly educated and knowledgeable about testing practices. The level of know-how of the project personnel is again directly linked to the testing culture of the organization. In addition, from a business perspective, the utilization of the testing practices from the more advanced maturity levels might not be viable if high testing coverage is not necessary.

As to discuss the test models, especially the test pyramid, in the light of the research, such a model is a great depiction of how the test quantities optimally divide in a highly automated testing setup. However, the research revealed that the project context could constraint the feasibility of test automation and its utilization on different test levels. Essentially, religious following of certain testing model might not be the best solution. Especially, as the focus areas could be different in backend and frontend. In addition, the inverted test pyramid could be the only viable option in some project and application contexts. Though, the research discussed that such a testing setup rarely leads to a quality product. This is due to the bulk

of testing effort being manual which, in turn, has a wide variety of issues. As such, the advancements in testing culture and maturity could enable to avoid the inverted pyramid.

5.3 Software development process consequences

The research results can be used in subsequent web application projects as guidance on testing considerations. The results do not advocate a certain universal truth for testing of web applications. Quite the opposite, as the results indicate that the best testing decisions are the ones that are considered within the organization, project and application context.

The heterogeneity of the unit's projects makes it difficult to standardize or mandate certain testing practices. However, especially when considering the supportive practices – there is a clear indication that high emphasis should be put on test planning and the practices of continuous integration and deployment. The research revealed that such practices are, in fact, quite accessible and implementable in a standardizable manner even in a heterogeneous project environment. In addition, the utilization of test automation should be cherished to enable truly continuous development and generally more mature testing practices.

As indicated by the research, web application development is swiftly moving forward. Therefore, it is essential to maintain an open mind and learn new ways to test the applications. Also, in such an environment, the knowledge transfer processes are more and more relevant. Sharing of the testing knowledge is beneficial on the organizational level and the cornerstone of testing culture.

5.4 Managerial implications

The interviewed personnel from the unit of the collaborating company discussed that the testing processes could be enhanced. The interviewees proposed that the testing effort and planning considerations could be further integrated into the sales and project initialization phase. As such, the testing considerations would be built-in into the process and subsequent

test resourcing, planning, implementation and monitoring would be streamlined. Also, some considered whether it would be beneficial to reinforce the unit's testing with a dedicated tester or test-oriented manager. Some proposed that the unit could advocate testing as a front-line practice in the future, in a similar fashion as currently with the user experience and service design.

The discussions led to conclusions that, such advancements in the field of web application testing could potentially be used as a competitive edge in the software consulting market. In addition, a solid foundation for testing practices could enable the unit to succeed and offer consultation in a wider range of projects. Such feats could, in turn, enable the unit's business to grow.

5.5 Research limitations

The primary limitation of the research is the fact that the research was limited to the context of one company. Therefore, the results of the thesis are company context-dependent and hardly generalizable. As such, the thesis focused on the developer viewpoint. In addition, the schedule of the research project further limited the extent of the use of different research methods. In hindsight, the development of a meaningful questionnaire was quite complex. On the other hand, the interviews were an efficient way to gather meaningful data as the discussions enabled to consider the researched phenomena from various viewpoints.

5.6 Future research directions

As the field of web application development is ever-changing, replication of similar research could be beneficial in the future. Mainly as the emergence of new revolutionary development technology, testing tool or methodology could fundamentally change the testing of web applications. Also, it would be interesting to investigate the topic in the context of other consulting companies to discuss whether there are contrasting views.

One interesting result of the research was that the testability of the code influences greatly the feasible testing methods. Future research could focus on how to design and write web applications in a way that enables efficient testing on all levels. Additionally, future research could focus on specific aspects of web application testing and discuss how to generalize the tools or implementation to heterogeneous projects.

6 CONCLUSION

The role of testing in modern-day web application development is significant. The definition of suitable testing practices for a web application project is fundamentally a complex problem. This is due to the fact that web application projects are diverse, web application testing is a complex field and various factors influence how the testing can and should be orchestrated in a certain project context.

Which project factors should be taken into account when considering the testing level of web application in an agile environment?

The research concludes, based on the occurrences and repetitiveness of the factors during the research conduction, that the project budget and schedule are the primary constraints on testing orchestration. In addition, the criticality and complexity of the application further highlight the need for functional and non-functional testing. Furthermore, the know-how of the project personnel is a primary driver on which testing practices are feasible and meaningful. Finally, various other factors were extracted during the research which further validates that the testing definition at a project level is a difficult task.

How to define sufficient testing level for web application projects relative to these project factors?

The research revealed that in an optimal situation, the functional testing is automated on all testing levels to the highest degree and manual testing is used only as a supplemental method for automated testing. In addition, the non-functional aspects of the application are tested. To achieve such a situation, the testing process must be ingrained to the software development practice and cherished by all parties. However, various factors, whether internal or external, contribute to the fact that rarely such a testing setup is feasible. To outline a sufficient testing effort for a certain web application project, skill and expertise are required. Decisions on how to utilize the available resources efficiently, regarding testing

within the product and project context, are critical. Fundamentally, the consideration focuses on which kind of risks the project has and how to get around the constraints that the project context introduces for the product quality.

As a general guideline, the research suggests considering, with great care, the risks that the lack or incorrect focus of testing could introduce. As such, it is more likely, that the possibly limited testing effort is directed to the right areas and a sufficient testing level is achieved. Utilization of a wide range of different testing practices enables critiquing the application from various dimensions. However, in the web application context, the research highlights system testing as a cost-effective testing method. It is found to be an efficient method to test the application functionally, it excels in the detection of regression and enables the detection of non-functional issues at a base level.

Additionally, especially in software consulting context, the advancements in testing culture could yield massive benefits. The capability of utilizing certain testing practices for the subsequent projects fundamentally originates from the testing culture and personnel know-how. Introduction of systematic testing for all projects further advocates the quality and testing culture. Test planning, employment of continuous integration and deployment as well as test automation are also highly recommended practices to advance in testing maturity. Fundamentally, advancements in testing culture enable the advancements in testing maturity. As such, the level of testing and subsequent product quality is increased regardless of the project or product context.

7 REFERENCES

1. Afzal, W., Ghazi, A. N., Itkonen, J., Torkar, R., Andrews, A. & Bhatti, K. 2015. An experiment on the effectiveness and efficiency of exploratory testing. Empirical Software Engineering, 20(3), pp. 844-878. doi:10.1007/s10664-014-9301-4
2. Agile Alliance 2019, Agile 101 [Accessed 2019-11-14] https://www.agilealliance.org/agile101/
3. Agile Manifesto 2001, Manifesto for Agile Software Development [Accessed 2019-11-14] https://agilemanifesto.org/
4. Albrecht, J. & Wadlinger, K. 2017, An Evaluation of Cloud-based Platforms for Web-Application Development, 12th International Conference on Software Technologies
5. Alenezi, M. & Javed, Y. 2016, Open Source Web Application Security: A Static Analysis Approach, 2016 International Conference on Engineering & MIS (ICEMIS)
6. Ammann, P. & Offutt, J. 2016, Introduction to Software Testing, Cambridge University Press, 2nd Edition
7. Android Developers 2019 Testing Fundamentals [Accessed 2020-01-31] https://developer.android.com/training/testing/fundamentals
8. Arora, A. & Sinha, M. 2012, Web Application Testing: A Review on Techniques, Tools and State of Art, International Journal of Scientific & Engineering Research, Vol. 3 (2)
9. Beizer, B. 1990, Software Testing Techniques, 2nd Edition
10. Black, R. 2009, Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3rd Edition, Wiley
11. Borle, N., Feghhi, M., Stroulia, E., Greiner, R. & Hindle, A. 2018. Analyzing the effects of test driven development in GitHub. Empirical Software Engineering, 23(4), pp. 1931-1958. doi:10.1007/s10664-017-9576-3
12. Brandon, D. 2008, Software Engineering for Modern Web Applications: Methodologies and Technologies, Hershey, PA

13. Casteleyn, S., Daniel, F., Dolog, P., Matera, M. 2009, Engineering Web Applications, Springer
14. Causevic, A., Sundmark, D. & Punnekkat, S. 2011. Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review. 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011; Berlin; 21 March 2011 through 25 March 2011, pp. 337-346. doi:10.1109/ICST.2011.19
15. Chemuturi, M. 2011. Mastering software quality assurance: Best practices, tools and techniques for software developers. Fort Lauderdale, Fla.: J. Ross Pub.
16. Clarke, P. & O'Connor, R. V. 2012. The situational factors that affect the software development process: Towards a comprehensive reference framework. Information and Software Technology, 54(5), pp. 433-447. doi:10.1016/j.infsof.2011.12.003
17. Cohen, L., Manion, L. & Morrison, K. 2007, Research methods in education, 6th edition, Routledge
18. Cohn, M. 2009. Succeeding with agile: Software development using Scrum. Upper Saddle River, NJ: Addison-Wesley.
19. Contan, A., Dehelean, C. & Miclea, L. 2018. Test automation pyramid from theory to practice, *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 24-26 May 2018
20. Crispin, L. & Gregory, J. 2009, Agile Testing: A Practical Guide for Testers and Agile Teams, Addison-Wesley
21. Di Lucca, G. A. & Fasolino, A. R. 2006. Web Application Testing, Web Engineering, pp. 219-260, Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg
22. Douglas, B, P. 2016, Agile System Engineering, BCS Learning & Development Ltd
23. Douyle, B. & Lopes, C. V. 2017, Survey of Technologies for Web Application Development
24. Drury-Grogan, M. L., Conboy, K. & Acton, T. 2017. Examining decision characteristics & challenges for agile software development. The Journal of Systems & Software, 131, pp. 248-265. doi:10.1016/j.jss.2017.06.003

25. Duskis, M. 2019, Waiter, There's a Database in My Unit Test!, [Accessed 2019-12-08] https://medium.com/better-programming/waiter-theres-a-database-in-my-unit-test-9698d866102e
26. Fasolino, A., Amalfitano, D. & Tramontana, P. 2013. Web application testing in fifteen years of WSE. Proceedings of IEEE International Symposium on Web Systems Evolution, WSE, pp. 35-38. doi:10.1109/WSE.2013.6642414
27. Fowler, M. 2006, Continuous Integration [Accessed 2019-12-13] https://martinfowler.com/articles/continuousIntegration.html
28. Fowler, M. 2012, Test Pyramid [Accessed 2019-12-08] https://martinfowler.com/bliki/TestPyramid.html
29. Fowler, M. 2018, Integration Test [Accessed 2019-12-15] https://martinfowler.com/bliki/IntegrationTest.html
30. Front-End Checklist 2019, GitHub: thedaviddias [Accessed 2019-12-27] https://github.com/thedaviddias/Front-End-Checklist
31. Ghazi, A. N., Petersen, K. & Börstler, J. 2015. Heterogeneous Systems Testing Techniques: An Exploratory Survey. 7th International Conference on Software Quality Days, SWQD 2015; Vienna; Austria, 200, pp. 67-85. doi:10.1007/978-3-319-13251-8_5
32. Goodman, D. 2018, Mendix, Agile Process: Why You Need Feedback Loops Both During and After Sprints [Accessed 2019-11-22] https://www.mendix.com/blog/agile-process-why-you-need-feedback-loops-both-during-and-after-sprints/
33. Graham, D., Veenendaal, V., Evans, I. & Black, R. 2008, Foundations of Software Testing: ISTQB Certification, Cengage Learning Emea
34. Grigera, J., Garrido, A., Rivero, J. M. & Rossi, G. 2017. Automatic detection of usability smells in web applications. International Journal of Human - Computer Studies, 97, pp. 129-148. doi:10.1016/j.ijhcs.2016.09.009
35. Hambling, B. 2010, Software Testing – An ISTQB-ISEB Foundation Guide (2nd Edition), BCS The Chartered Institute for IT

36. Hlebowitsh, N. 2019 Tecla: 2019 Stats on Top JS Frameworks: React, Angular & Vue [Accessed 2019-11-22] https://www.tecla.io/blog/2019-stats-on-top-js-frameworks-react-angular-and-vue/
37. iClerisy 2019, An Introduction to Continuous Integration (CI) and its Benefits [Accessed 2019-12-13] https://iclerisy.com/what-is-continuous-integration-ci/
38. ISO/IEC 25010:2011, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models
39. Itkonen, J. & Mäntylä, M. 2014. Are test cases needed? Replicated comparison between exploratory and test-case-based software testing. Empirical Software Engineering, 19(2), pp. 303-342. doi:10.1007/s10664-013-9266-8
40. Jest 2019, Jest Docs, [Accessed 2019-12-07] https://jestjs.io/en/
41. JReport 2019, 3-Tier Architecture: A Complete Overview [Accessed 2019-11-29] https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/
42. Kaluža, M., Kalanj, M, Vukelić, B. 2019, A Comparison of Back-end Frameworks for Web Application Development, Zbornik Veleučilišta u Rijeci 2019, Vol. 7 (1), pp. 317-332
43. Kasurinen, J., Taipale, O. & Smolander, K. 2010. Software Test Automation in Practice: Empirical Observations. Advances in Software Engineering.
44. Kasurinen, J. 2012, Software Organizations and Test Process Development, Advances in Computers, Vol. 85, pp. 1-63
45. Kappel, G. 2006, Web Engineering: The Discipline of Systematic Development of Web Applications, John Wiley & Sons Ltd
46. Karac, I. & Turhan, B. 2018. What Do We (Really) Know about Test-Driven Development? IEEE Software, 35(4), pp. 81-85. doi:10.1109/MS.2018.2801554
47. Khvan, X. 2017. RedmineUP, DevOps in Redmine [Accessed 2019-12-29] https://www.redmineup.com/pages/blog/devops-in-redmine
48. Kortelainen, H., Happonen, A., 2017, Digitalisaatio muokkaa tiedon hallintaan pohjautuvien palveluiden markkinoita, In VTT Blog, doi: 10.13140/RG.2.2.10075.52006

49. Kyryk, I. 2018, QATestLab: What projects need test automation [Accessed 2019-11-22] https://blog.qatestlab.com/2018/06/12/when-automate-testing/
50. Laine, M., Shestakov, D., Litvinova, E & Vuorimaa, P. 2011, Toward Unified Web Application Development, IT Professional September 2011, Vol. 13 (5), pp. 30-36
51. Leotta, M., Clerissi, D., Ricca, F. & Tonella, P. 2013. Capture-replay vs. programmable web testing: An empirical assessment during test case evolution.
52. Loadster 2019, Front-End vs Back-End Performance [Accessed 2019-12-27] https://loadster.app/articles/front-end-vs-back-end-performance/
53. Mark, S. 2007, Test-Driven Development: An Agile Practice to Ensure Quality is Built from the Beginning, Agile Software Development Quality Assurance, Chapter 11, Hershey PA: Idea Group Reference
54. Matam, S. K. & Jain, J. K. 2017. Pro Apache JMeter: Web Application Performance Testing. Berkeley, CA: Apress.
55. Measey, P. 2015, Agile Foundation: Principles, Practices and Frameworks, Elsevier Inc
56. Mertz, J. & Nunes, I. 2018. Understanding Application-Level Caching in Web Applications: A Comprehensive Introduction and Survey of State-of-the-Art Approaches. *ACM Computing Surveys (CSUR), 50*(6), pp. 1-34. doi:10.1145/3145813
57. Meyer, M. 2014. Continuous Integration and Its Tools. IEEE Software, 31(3), pp. 14-16. doi:10.1109/MS.2014.58
58. Meyer, M. A. & Booker J. M. 2001, Elicitating and Analyzing Expert Judgment: A Practical Guide, Academic Press Limited
59. Mimick, 2014. Continuous Delivery Maturity Model, [Accessed 2020-02-02] https://developer.ibm.com/urbancode/docs/continuous-delivery-maturity-model/
60. Mok, W., Hickman, C. & Allport, C. 2013. Implementing Business Processes: A Database Trigger Approach. International Journal of Knowledge - Based Organizations, 3(2), p. 36. doi:10.4018/ijkbo.2013040103
61. Mårtensson, T., Ståhl, D. & Bosch, J. 2017. Exploratory testing of large-scale systems – Testing in the continuous integration and delivery pipeline. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and

Lecture Notes in Bioinformatics), 10611, pp. 368-384. doi:10.1007/978-3-319-69926-4_26

62. Mohagheghi, P. 2008. Evaluating software development methodologies based on their practices and promises, New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Seventh SoMeT 2008, October 15-17, United Arab Emirates
63. Murugesan, S. 2008, Web Application Development: Challenges and the Role of the Web Engineering, Web Engineering: Modelling and Implementing Web Applications, pp. 7-32
64. Mäkinen, S., Puonti, M., Lehtonen, T., Mikkonen, T., Kilamo, T. & Männistö, T. 2019. Revisiting continuous deployment maturity: A two-year perspective.
65. O'Connor, R., Elger, P. & Clarke, P. M. 2016. Exploring the Impact of Situational Context - A Case Study of a Software Development Process for a Microservices Architecture.
66. OWASP 2015. OWASP Testing Guide v4 [Accessed 2019-12-21] https://www.owasp.org/images/1/19/OTGv4.pdf
67. OWASP 2017. OWASP Top 10 – 2017 [Accessed 2019-12-21] https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
68. Parzych, D. 2016, Catchpoint, Front-end vs Back-end vs Network Performance [Accessed 2019-12-27] https://blog.catchpoint.com/2016/11/03/naming-conventions/
69. Patton, R. 2005, Software Testing (2nd Edition), Sams Publishing
70. Rehn, A., Palmborg, T. & Boström, P. 2014, The Continuous Delivery Maturity Model [Accessed 2020-02-02] https://www.infoq.com/articles/Continuous-Delivery-Maturity-Model/
71. Scott, Testing Pyramids & Ice-Cream Cones [Accessed 2019-12-08] https://watirmelon.blog/testing-pyramids/
72. Stack Overflow 2019, Developer Survey 2019 [Accessed 2019-12-14] https://insights.stackoverflow.com/survey/2019

73. Sun, Y. 2019, Practical Application Development with AppRun: Building Reliable, High-Performance Web Apps using Elm-Inspired Architecture, Event Pub-Sub and Components, Apress
74. StackShare 2019. Airbnb. [Accessed 2019-12-05] https://stackshare.io/airbnb/airbnb
75. State of Agile 2019, 13th Annual State of Agile Report, CollabNet VersionOne [Accessed 2019-11-14]
76. State of CSS 2019, State of CSS [Accessed 2019-11-22] https://2019.stateofcss.com/
77. State of Testing 2019, State of Testing Report, PractiTest [Accessed 2020-01-24] https://qablog.practitest.com/state-of-testing/
78. Ståhl, D. & Bosch, J. 2014. Modeling continuous integration practice differences in industry software development. The Journal of Systems & Software, 87(1), pp. 48-59. doi:10.1016/j.jss.2013.08.032
79. SWEBOK 2014, Bourque, P. & Fairley R.E. Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014, https://www.swebok.org
80. Tarlinder, A. 2016, Developer Testing: Building Quality into Software, Addison-Wesley Professional
81. Tikhanski, D. 2018, BlazeMeter: Open Source Load Testing Tools: Which One Should You Use? [Accessed 2019-12-27] https://www.blazemeter.com/blog/open-source-load-testing-tools-which-one-should-you-use/
82. TMMi Foundation 2019, TMMi Model, [Accessed 2020-02-06] https://www.tmmi.org/tmmi-model/
83. Toivanen, I. 2019, Visma Consulting, DevOps jatkuvan kehittämisen tukena [Accessed 2019-12-29] https://www.vismaconsulting.fi/blogi/devops-jatkuvan-kehittamisen-tukena
84. Torchiano, M., Ricca, F. & Marchetto, A. 2011, Are web applications more defect-prone than desktop applications?, *International Journal on Software Tools for Technology Transfer*, 13 (2), pp. 151-166. doi:10.1007/s10009-010-0182-6
85. Trobia, A. 2008, Encyclopedia of Survey Research Methods, Encyclopedia of survey research methods. Los Angeles, [Calif.] London: SAGE.

86. tSQLt 2019, Test Driven Design for SQL Server, [Accessed 2019-12-07] https://tsqlt.org/
87. Web Developer Roadmap 2019, GitHub: kamranahmedse [Accessed 2019-11-22] https://github.com/kamranahmedse/developer-roadmap
88. Van Den Broek, R., Bonsangue, M.M, Chaudron, M. & Van Merode, H. 2014, Integrating Testing into Agile Software Development Processes, Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development
89. Veenendaal, E. V. 2019, TMMi in the Agile world, Version 1.3, TMMi Foundation
90. Vemula, R. 2017, Real-Time Web Application Development: With ASP.NET Core, SignalR, Docker and Azure, Apres
91. Vocke, H. 2018, The Practical Test Pyramid, [Accessed 2019-12-07] https://martinfowler.com/articles/practical-test-pyramid.html

APPENDIX 1. Translated questionnaire

Title: Testing decisions and web-applications

Introduction:

The following questionnaire is a part of a Master's Thesis which investigates how to define suitable testing levels, methods and supportive practices for an agile web application project. The target of the questionnaire is to identify the most significant contextual project factors that affect testing decisions.

Demographic questions:

Current role in the organization:

- Superior / Manager
- Project manager
- Architect / Specialist
- Developer
- UX

Work experience in software development:

- 0-2 years
- 3-5 years
- 6-10 years
- 11-15 years
- 16-20 years
- +20 years

Work experience in web-application projects:

- None
- 0-2 years
- 3-5 years
- 6-10 years
- 11-15 years
- 16-20 years
- +20 years

Main question title:

In a free format, list 5-10 factors that in your view have the highest significance on the testing decisions of a web application project

Main question description:

You can utilize the following high-level categories for the factors, if you choose to (Clarke et al. 2012):

Application - Characteristics of the application(s) under development
Business - Strategic and tactical business considerations
Management - Constitution and characteristics of the software development management team
Organisation - Profile of the organisation
Operation - Operational considerations and constraints
Personnel - Constitution and characteristics of the non-managerial personnel involved in the software development efforts
Requirements - Characteristics of the requirements
Technology - Profile of the technology being used for the software development effort

APPENDIX 2. Translated interview structure

Introduction:

- Research background
- Research ethicality
- Demographic information

Themes:

Significance of testing

- In your view, how significant is testing of web applications?
- Why is the testing of web applications significant/insignificant?
- Should testing be a part of every web application project?
- How do you find the overall testing effort requirement in web application projects?

Testing levels

- In your view, is/should some testing level be highlighted in the testing of web applications?
- How would you divide the testing effort for the different testing levels?

Automated vs manual system testing

- In your view, how significant is the utilization of automated system testing in web application projects?
- Why is the utilization of automated system testing significant/insignificant?
- Do you find automated or manual system testing more beneficial?
- What kind of challenges are in automated and manual system testing?
- In which kind of web application projects, the utilization of automated system testing is beneficial?

Non-functional testing

- How significant is the non-functional testing (e.g. performance, security and usability testing) of web applications?
- Why is non-functional testing significant/insignificant?
- Which non-functional quality attributes of web applications require the most testing?
- What kind of challenges are in the adoption of non-functional testing?
- In which kind of projects the non-functional testing is beneficial?

Supportive practices

- How significant is continuous integration and delivery in web application development?
- In which kind of projects the utilization of continuous integration and delivery is beneficial?
- How significant is the utilization of test-driven development in web application development?
- Are there any other practices that support testing in web application development?

Project factors

- How does the project budget affect the testing of web applications?
- How does the project/application criticality affect testing of web applications?
- How does the project schedule affect the testing of web applications?
- How does the know-how of project personnel affect the testing of web applications?
- How does technology (availability and limitations) affect the testing of web applications?