DEGREE PROJECT IN THE FIELD OF TECHNOLOGY
COMPUTER SCIENCE AND ENGINEERING
AND THE MAIN FIELD OF STUDY
INDUSTRIAL MANAGEMENT,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Technical decision-making in startups and its impact on growth and technical debt

CARL HULTBERG

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Technical decision-making in startups and its impact on growth and technical debt

by

Carl Hultberg

Master of Science Thesis TRITA-ITM-EX 2021:240
KTH Industrial Engineering and Management
Industrial Management
SE-100 44 STOCKHOLM

Tekniskt beslutsfattande i startups och dess påverkan på tillväxt och teknisk skuld

av

Carl Hultberg

Examensarbete TRITA-ITM-EX 2021:240
KTH Industriell teknik och management
Industriell ekonomi och organisation
SE-100 44 STOCKHOLM

KTH Industriell teknik
och management

Master of Science Thesis TRITA-ITM-EX 2021:240

Technical decision-making in startups and its impact on growth and technical debt

Carl Hultberg

Approved	Examiner	Supervisor
2021-06-02	Matti Kaulio	Anna Jerbrant

Abstract

The rapid pace of digitalization has resulted in increased management of software development, and today a majority of startups are reliant on software. How to manage software development projects is a well-researched area and agile methods are widely adopted by companies in all industries and sizes. However, prior to working with agile methods or any other software development methodology, the founders and management of a startup have to make several technical decisions that could potentially affect the whole software development process and the company's success. Furthermore, studies show that only three programming languages are known by more than 50% of developers, suggesting that the potential effects of technical decisions stretch outside the software development process.

By performing a multiple-case study on startups with a mixed-methodology approach, the researcher has analyzed the literature, interviewed several founders and Chief Technology Officers, and quantitatively analyzed hundreds of thousand lines of code, to find how to organize to make better technical decisions in order to enhance growth and generate less technical debt. The results show that the effects of technical decisions stretch outside the software development process, having an apparent effect on a startup's ability to attract and retain talent. Furthermore, the results show that access to talent is an important but not deciding factor in technical decision-making. Additionally, it is found that in the initial stage of a startup, ease of development and speed are important factors in technical decisions as the main objective is to find product-market fit. When product-market fit has been found and the startup matures, the focus shifts and quality and durability are becoming prominent factors. It is found that scooping features only to implement the absolute core functionality is an effective approach to develop quickly and generate less technical debt while maintaining customer satisfaction. Lastly, it is found that programming language affects the number of issues generated per line of code and the time spent on building features. However, as found in the literature, there is no evidence of this being related to the type of programming language.

The findings have both practical and academic implications. In academics, this thesis lays the foundation for further studies and provides new insights into the field of startups in general, and technical decision-making in particular. For practitioners, this thesis provides a basis for discussion and execution of technical decisions in the early stages of a startup.

Key-words

Technical Decisions, Startups, Technical Debt, Talent Management, Technical Stack, Programming Languages, Complexity, Software Development

KTH Industrial Engineering and Management

Examensarbete TRITA-ITM-EX 2021:240

Tekniskt beslutsfattande i startups och dess påverkan på tillväxt och teknisk skuld

Carl Hultberg

Godkänt	Examinator	Handledare
2021-06-02	Matti Kaulio	Anna Jerbrant

Sammanfattning

Den snabba digitaliseringen har resulterat i en ökad ledning av mjukvaruutveckling och idag är majoriteten av startups beroende av någon form av mjukvara. Hur man leder mjukvaruutvecklingsprojekt är ett välutforskat område och agila metoder är välanvända i företag i alla industrier och storlekar. Innan man arbetar med agila metoder eller någon annan mjukvarutvecklingsmetod så måste grundarna och ledningen ta flera tekniska beslut som potentiellt kan påverka hela mjukvaruutvecklingsprocessen och företagets framgång. Samtidigt finns det studier som visar att endast tre programmeringsspråk hanteras av mer än 50% av utvecklarna, vilket indikerar att de potentiella effekterna av tekniska beslut sträcker sig långt utanför mjukvaruutvecklingsprocessen.

Genom att utföra en flerfallsstudie på startups med både kvalitativa och kvantitativa moment, har forskaren analyserat literaturen, intervjuat flertalet grundare och tekniska chefer, och kvantitativt analyserat hundratusentals rader kod, för att undersöka hur startups kan organisera sig för att ta bättre tekniska beslut som förbättrar tillväxten samt genererar mindre teknisk skuld. Resultaten visar att effekten av tekniska beslut sträcker sig långt utanför mjukvaruutvecklingsprocessen genom att ha en direkt påverkan på startups möjlighet att attrahera och behålla talang. Tillgången till talang visar sig även vara en viktig faktor i teknisk beslutsfattande, däremot är den inte en avgörande faktor. Dessutom visar resultaten att i det initiala stadiet av en startup så är enkelhet och hastighet viktiga faktorer i tekniskt beslutsfattande eftersom fokus ligger på att hitta produkt-marknads-anpassning. När produkt-marknads-anpassning är funnen och startupen mognar, så skiftar dessa faktorerna över till kvalité och hållbarhet. Resultaten visar även att en effektiv metod för att utveckla snabbt och skapa mindre teknisk skuld är att skala ner förfrågningar till dess absolut grundfunktionalitet, samtidigt visade det sig att kundnöjdheten inte minskade. Slutligen visar resultaten att val av programmeringsspråk har en effekt på antalet issues genererade per rad kod och även tiden spenderad för att bygga features. Däremot, precis som i tidigare forskning, finns det inga bevis på att det är relaterat till typen av programmeringsspråk.

Resultaten har både praktiska och akademiska implikationer. I den akademiska världen så lägger detta arbetet en grund för framtida forskning och ger nya insikter i startupfältet generellt, och tekniskt beslutsfattande i startups i synnerhet. För utövare, lägger detta arbetet en bra bas för diskussion och verkställande av tekniska beslut i startups.

Nyckelord

Tekniska beslut, Startups, Teknisk skuld, Talanghantering, Teknisk stack, Programmeringsspråk, Komplexitet, mjukvaruutveckling.

Acknowledgements

I would like to thank my thesis supervisor, Anna Jerbrant, for invaluable feedback and interesting discussions throughout the process of writing this thesis. Furthermore, I would like to extend my gratitude to Jens Bäckbom at Almi Invest, and Fabian Assarsson at Hidden Dreams, for being my sounding board and helping me get in contact with prominent entrepreneurs that partook in this thesis. Lastly, I would like to thank the aforementioned entrepreneurs for investing their time and sharing their thoughts and data on the topic of this thesis.

Stockholm, June 2021

Carl Hultberg

Acronyms

TM	Talent Management
LOC	Lines of Code
CTO	Chief Technical Officer
CPO	Chief Product Officer
CEO	Chief Executive Officer
B2B	Business to Business
B2C	Business to Customer
MVP	Minimum Viable Product
ACE	Alignment, Capabilities, Engagement
SME	Small- and Medium Enterprise
AI	Artificial Intelligence
ML	Machine Learning

Contents

1 Introduction

In this chapter, aims to introduce the topic of technical decision making, talent management and organizational structures in startups.

1.1 Background

For the last decade the digitalization has continued to increase rapidly and as expressed by venture capitalist Marc Andreessen in 2011, "software is eating the world" (McKinsey & Company 2016). Digitalization has resulted in many internal projects within organizations, but it has also created business opportunities for startups. With the increased focus on digitalization, almost all startups involve some software development (McKendrick 2017). Managing software development projects is a well-researched area, and many methods such as SCRUM and Waterfall are widely adopted by companies in all industries conducting software development. However, prior to working with agile methods, or any other software development method, the founders of a startup have to make several decisions that could potentially affect the whole software development process and the company's success. One of these decisions is choosing the technology stack. A technical stack refers to the technical infrastructure of a service or program. More precisely, it refers to the list of services that are required in order for the service or program to be functional. When discussing technology in companies, a phrase commonly used by software engineers is technical debt, which generally refers to the implied cost of any additional work required due to choosing the easy way that results in a limited solution, in contrast to taking the time with a better approach.

There has long been a common perception that programming paradigms such as

functional programming leads to fewer bugs. However, there are only a few studies that investigate this, and none of them is focusing on the startup phase of the company where speed is crucial (Ray et al. 2014). Neither is there a study looking at the relationship between technology stacks and successful software development. In many organizations, the chosen technology stack often has its basis in the stack that has been used priorly by the organization. Apart from the obvious factor of convenience in terms of integrations and knowledge, there are no clear motivations for using the same technology stack when starting a new project or company. One reason could be that no studies are investigating how the technical stack could affect the success of the software development process. However, startups almost always start with a tabula rasa and can therefore choose the technology stack more freely. With the common conception that certain programming paradigms lead to fewer bugs in mind, could there be technology stacks that result in less time spent on fixing bugs, and more time building functionality, i.e., more value-adding activities?

Furthermore, in a recent survey from Stack Overflow, it is shown that there are only three programming languages that are used by more than 50% of the respondents. When looking at frameworks, none are used by more than 50% of the respondents (Stack Overflow 2020). These answers raise the question of how the choice of the technology stack affects a startup's ability to hire software developers, an already widespread issue in the startup ecosystem, and a vital factor of a startup's ability to develop and scale its business. Early-stage venture capital investors have often been quoted as saying that the most critical factor in the early stages of a startup is the management team. Building the early organization is, therefore, among the most critical tasks for founders. However, it requires significantly more effort than

just hiring candidates that seem to be qualified. An organization's requirements and demands will significantly change during the startup life-cycle, and therefore require meticulous planning and execution in order to ensure that the organizational structure and staffing are coordinated with the overall strategy and needs of the business (Picken 2017). According to Fombrun & Wally (1989), if a startup fails to do so, the result is likely a lack of critical skills when needed, key hires getting worn out, ineffective decision process, and a lack of accountability. As the journey of digitalization continues rapidly, the role of human capital in the future might be questioned since parts of human labor can soon be replaced with autonomous systems. However, while these discussions go viral in popular culture, there is a silent war going on where companies are fighting for the top talent in all areas. The book "In Search of Excellence" Peters & Waterman (1984) analyzed the key factors of success and high performing companies in the United States, where it was recognized that viewing a companies employees as a source of quality was one of the key indicators. A decade later, McKinsey employees Michaels et al. (2001) first mentioned the war on talent, arguing that company employees are the key factor of success and that there will be a shortage of talent which would put significant stress on companies. Moreover, they argued that a company's ability to attract, recruit and retain talent would be the deciding factor between a winning or failing company. Furthermore, Michaels et al. (2001) argues that the war on talent is not only about treating employees well. It is also of utmost importance to have the right employee, at the right place, at the right time, which is argued by Picken (2017) and Fombrun & Wally (1989) as one of the most common reasons for failure in a startup.

In a recent study from McKinsey built on refreshed data of Michaels et al. (2001)

study, it is shown that jobs of high complexity in terms of interaction and information, such as the work of managers and software developers, high-performance individuals are up to 8 times as productive as average performers. The study also shows that the difference in productivity between an average and high performer exponentially increases as the complexity increases (Keller & Meaney 2017).

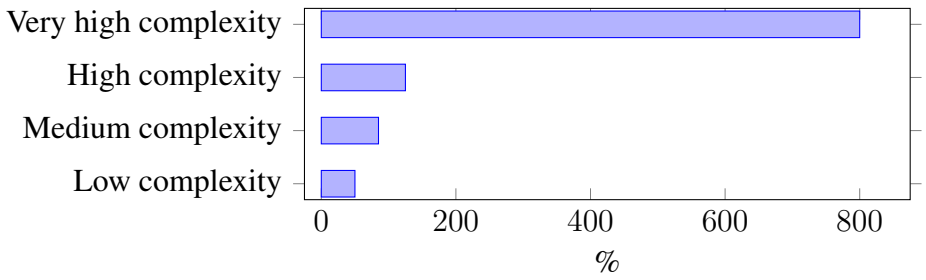

Figure 1: Productivity gap between average performers and high performers, by job complexity, (Michaels et al. 2001, Keller & Meaney 2017) (data refreshed by McKinsey 2012)

Additionally, the financial strength of startups is often significantly weaker than the one of established technology companies, which are all recruiting from similar talent pools. Established technology companies such as Apple, Google, and Shopify also have working talent management systems in place. Additionally, these companies have already invested years of work on employer branding, further complicating hiring efforts and increasing the competition for startups.

The aforementioned examples showcase the potential impact of early technical decisions in a startup, but how are these decisions taken and motivated? More precisely, how can startups be organized to make better and more informed technical decisions, and how do these decisions affect access to talent?

1.2 Problem formulation

With the rapid increase of digitalization and the increased management of software development in startups, understanding how to maximize these efforts is of utmost importance for founders and management in order to increase the odds for success. As access to technology is democratized and so the opportunity to build startups, quality and speed are becoming dominant factors in the success of startups. Concerning the high cost of software development, as much time as possible should be spent on building the software instead of fixing bugs. By taking the aforementioned common conception that some programming languages generate fewer bugs into consideration while also weighing in the small number of programming languages known by more than 50% of developers, the potential impact and importance of technical decisions becomes apparent. To better understand technical decision-making in startups, the factors behind the decisions need to be investigated and how the decisions affect the startup, both in terms of building the software and the ability to recruit and retain talent.

1.3 Purpose

This study aimed to get a better understanding of the technical decision-making process of a startup and how a startup can organize to make better technical decisions in order to enhance the growth and create less technical debt.

1.4 Research Question

Based on the background and purpose of the thesis, the following main research question (MRQ) is posed:

MRQ: *How should founders and management organize to make better technical decisions in the startup phase of a company in order to enhance growth and create less technical debt?*

To be able to answer the main research question, a subset of questions need to be answered. These are formulated as follows:

SRQ 1: *What are the factors of influence in a startup's technological decision making?*

SRQ 2: *What are the relationship between the choice of technology stack in general, and the programming language in particular, and the time spent on implementing features?*

SRQ 3: *How do technological decisions affect the ability to recruit and retain talent in startups?*

1.5 Delimitations

The study will be delimited to software startups in the Nordics. In order to be eligible for the study, companies need to fulfill the guidelines of what a small- and medium enterprise (SME) is according to (European Commission, 2014). Furthermore, several factors affect the growth of startups; this study is delimited to the effects of technical decisions.

2 Theoretical Framework

In this chapter, a comprehensive presentation of the key concepts and theories within the subject area of the thesis is presented. The purpose of this chapter is to create a theoretical framework of Talent Management, Employer Branding, Organization and Software Development in Startups, and Complexity analysis, that acted as a basis for the thesis.

2.1 Talent Management

Since talent management (TM) is a loosely defined term, Lewis & Heckman (2006) provides us with three different perspectives of TM. Firstly, practitioners with a specific focus within human resources often narrow down the definition of TM. For example, recruiters often narrow it down to sourcing and training the best talent when discussing TM. In contrast, compensation experts often narrow it down to a compensation and performance management process (Garger 1999, Cohn et al. 2005). Secondly, Lewis & Heckman (2006) mentions the perspective that TM mainly focuses on talent pools which are also discussed by Collings & Mellahi (2009). The creation of talent pools aims to create a flow of talents to fill open positions within the organization and is often referred to as succession planning in the literature. However, it can also include the recruitment and selection process (Rothwell 2010). According to Lewis & Heckman (2006), a vital role in this perspective is to project the needs of the business in terms of staffing and overseeing how employees grow within the organization. Finally, the last presented perspective of TM focuses on talent in general without considering the organization. In this perspective, talent is defined as high-performing individuals who are sought

for in organizations and not tied to a specific role. This stands in contrast to the second perspective, as the created talent pool is more general rather than building a specific talent pool to manage succession in the organization. Within this perspective, the same discussion as between Smart (1999) who argues that the whole organization should be filled with "A-players" and Michaels et al. (2001) who argues for differentiating the employees in performance categories, from which Collings & Mellahi (2009) adds another dimension by also differentiating the roles in terms of strategic impact within the organization.

The state of the TM literature has developed during the last two decades as described by Thunnissen & Gallardo-Gallardo (2017). With a significant increase in published articles in the last decades, Gallardo-Gallardo et al. (2015) argues that the TM research area is showing signs of moving to a more mature state, although not there yet. Collings et al. (2015) builds on a similar point, arguing for the increased legitimacy for TM as a research area, but, on the other hand, also arguing that the lack of clarity surrounding the conceptional and theoretical boundaries of the subject is one of the primary sources for the research area's skepticism. Another concern raised by Collings et al. (2015) is that some areas which are clearly within the field of talent management are not framed in the language of talent management and are therefore excluded from literature reviews, which may result in an underestimation of the work in the area. Dries (2013) has argued that the research agenda in the TM field is driven by phenomenon rather than theory. One potential explanation of this could be found in the arguments as mentioned earlier of Peters & Waterman (1984) and Michaels et al. (2001) who states that one of the biggest challenges for companies is the management of talent, from

which TM in practice has emerged. Another result of the underdeveloped nature of the TM field is the number of different definitions of TM, and as argued by Ashton & Morton (2005) "... is not a single consistent or concise definition". However, the most well-cited definition is from Collings & Mellahi (2009) and they argue that the starting point of any talent management system is the systematic identification of key positions within an organization that differentially contribute to the organization's competitive advantage. When identifying the key positions,

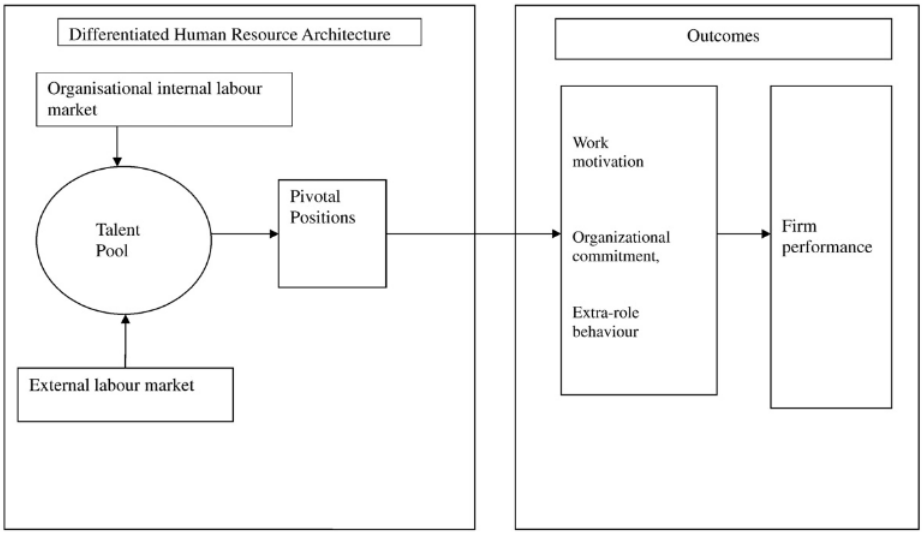

Figure 2: Talent Management system. In the system above, a pivotal role is the same as a strategic role as presented in the framework (Collings & Mellahi, 2009)

Collings & Mellahi (2009) follows Becker & Huselid (2006) in recognizing that there should be a differentiation between strategic and non-strategic roles within an organization and in this case, a greater focus on strategic roles. In other words, identifying key roles is aligned with the potential impact the roles can have on the organization. Furthermore, they emphasize the importance of creating a pool of high-performing candidates and have high potential to fill the key position

discussed in the first part of their definition. The focus on the key positions with high impact on the organization stands in contrast to previous definitions such as from Smart (1999) arguing that every role in the organization should be filled with high-performers. With the differentiation of roles, Collings & Mellahi (2009) says that "It is neither practical nor desirable to fill all positions in an organization with A performers" and argues that only focusing TM on key roles would facilitate better usage of the organization's resources. Lastly, once talent is identified and attracted to the organization, the work to maximize the efforts received from them through appropriate human resource policies begins. This whole process is shown in Figure 2 above.

The TM process can be divided into three categories: attract, develop, and retain. This division of the TM process has been used by several authors such as Stahl et al. (2016), Hiltrop (1999) and Oladapo (2014) among others. These three categories are used in several other frameworks, such as the "Talent Lifecycle" presented by Schiemann (2014). The talent lifecycle shown in Figure 3, is a bit more detailed in its stages and touches upon all interaction areas between a company and its human capital. However, to get a better overview, the following parts of the literature review within the field of TM will be divided according to attract, develop and retain as these are prevalent keywords in the literature.

2.1.1 Attract

In Stahl et al. (2016) 's article, they research how 37 multinational companies sustain their talent pipeline. The companies put a strong emphasis on "high potentials" in their TM work which is aligned with the definition from Collings

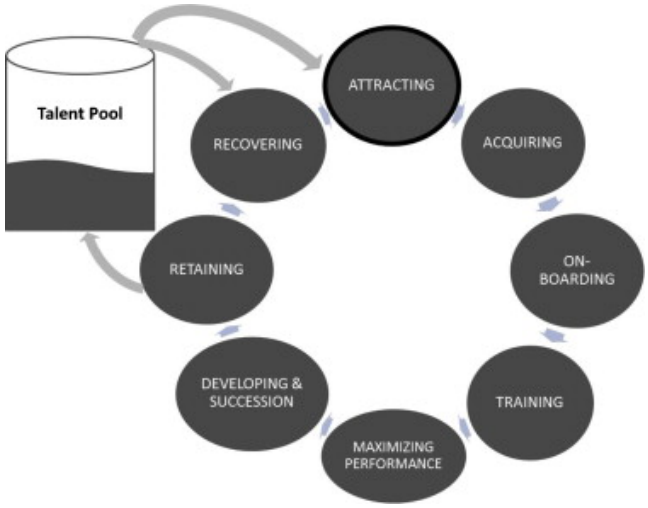

Figure 3: The talent lifecycle (Schiemann 2014)

& Mellahi (2009). In the hunt for talent, most companies follow a similar path by trying to recruit the best people who then are placed within the company rather than recruiting specific candidates for specific positions. The companies in Stahl et al. (2016) study recruit talent through a broad selection of channels such as the internet and on-campus events. However, in order to attract talent, most companies try to develop a relationship with the top universities of the world. Stahl et al. (2016) adds that to attract the best candidates, companies need to put much effort into branding. In other words, companies need to see candidates as potential customers and analyze them in order to find which attributes matters the most for the candidates and how to reach them. In older literature, the most common advice given in order to attract talent is to treat and pay talent well (Cappelli 1999). However, several studies conclude that salary does not have a significant effect on attracting talent in many countries and industries (Hiltrop 1999). This fact is also supported by Michaels et al. (2001) who argues that financial incentives are

important. However, it cannot replace a clear career path, exciting challenges, and mentorship from senior employees. Michaels et al. (2001) refers to this as creating and delivering an attractive "employee value proposition". A recent survey focused on early-career technical talent shows that the most sought-for perk that would attract the respondents to an organization is "Matching employee 401k contributions", "Quality of healthcare plan", and "student loan repayment". Interestingly, when divided into subgroups of age 22-25, 26-30, 31-35, and 36+, the ranking of perks is the same except for "Student loan repayment" for 36+. Thus, the study suggests that the millennial generation might not be so different from earlier generations in terms of what is attracting them to join an organization (Zaharee et al. 2018).

2.1.2 Develop

Once talent has been attracted to an organization, the question of nurturing and developing the talent accrued in the organization arises. In the 50s internal development of talent was the norm, which continued for several decades (Cappelli 2013). The way companies applying internal development of talent is very similar to succession planning as described by Collings & Mellahi (2009). However, Cappelli (2013) argues that succession planning and internal development of talent is risky, slow, and outdated as the world has become more dynamic. Cappelli (2013) argues that TM should follow the leads of supply chain management, which is outsourced (i.e., recruiting talent externally, not developing internally) but also mentions that this creates uncertainty whether the talent will be available when needed. Schuler et al. (2011) takes another perspective and argues that internal training and development can be used well in situations where the sought-for com-

petencies are of low supply, which at the same time can increase the organization's appeal to external talent. There is a discussion both in academics and in practice regarding how to distinguish the need to invest in the management of talent versus the need to invest in the selection process of talent versus the need to invest in the development and training of talent (Lewis & Heckman 2006). With the HC Bridge Decision Framework (Figure 4) Boudreau & Ramstad (2005) put some color into this discussion. The model applies three levels of analysis: impact, effectiveness, and efficiency. In the model, impact refers to the strategic effects of a change in the talent pool, more precisely, how a change in performance and depth would help the organization reach its strategic goals. Similar to Collings & Mellahi (2009), Boudreau & Ramstad (2005) puts emphasis on the segmentation (Collings & Mellahi uses the word "differentiation") of the talent pool, where jobs of high strategic importance are more important to impact. Effectiveness refers to how active changes affect the talent pool. The effects of these changes are independent of impact, and therefore changes motivated by effectiveness must be done with regard to impact. For example, spending time developing and training a talent pool with low strategic importance will have little impact on the overall strategy as the improved performance does not impact the execution. Lastly, efficiency refers to how much an investment generated in some measurement which is commonly used in HR practices such as cost per hire. Boudreau & Ramstad (2005) emphasizes the irrelevance of efficiency measures when effectiveness and impact measures are absent.

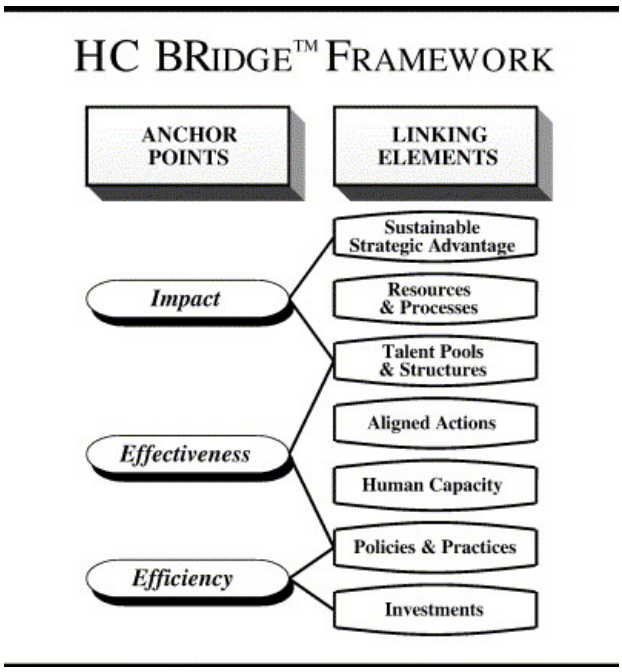

Figure 4: HC Bridge Decision Framework (Boudreau & Ramstad 2005)

2.1.3 Retain

In much of the literature, retention is discussed alongside the attraction and therefore, the arguments and discussions in the literature are similar. For instance, the "employee value proposition" which is proposed by Michaels et al. (2001) is also applicable in the case of retaining talent. Hiltrop (1999) study looks into which HR practices have the biggest effect on the attraction and retention of talent. Even though the results of the effect being significant, the proportion was low enough to suggest that other factors largely influence the talent. However, the study showed that bigger companies were better at attracting and retaining talent than small and medium-sized companies. This could be explained by Stahl et al. (2016) who argues that the best companies in terms of attraction and retention put much effort

in building their brand among talent, an activity that requires significant effort and financial strength. Schiemann (2014) discusses the common conception that the majority of departing employees are due to a bad relationship with the direct report. However, as found by Hiltrop (1999), Schiemann argues that there are several factors involved in most cases that make it harder for management to improve retention. While much effort in both the literature and by practitioners is put on the pivotal roles of a company (Michaels et al. 2001, Hiltrop 1999, Stahl et al. 2016, Collings & Mellahi 2009, Cappelli 1999), e.g., the roles where better or more talent will have the highest strategic impact, the retention in these roles are are a significant challenge for organizations (Boudreau & Ramstad 2007, Cascio & Boudreau 2010). To add some color to the issue of having several factors of departure, Schiemann (2014) mentions the People Equity frameworks (Figure 5) key areas: alignment, capabilities, and engagement (ACE). Alignment refers to how employees are aligned with the business strategy, brand values, and customers. Capabilities refer to how well an organization can create and develop knowledge, resources, and information. Lastly, engagement refers to how engaged the employees are to the organization and its values, in essence, how well the employees advocate for the organization. Schiemann argues for the importance of both measuring ACE as well as measuring the drivers of ACE, as it is possible to find a link between those measurements and low retention of employees.

2.1.4 In startups

There is not much academic literature focusing on talent management in startups. However, there is substantial grey literature from industry experts, venture capital investors, and startup founders on the topic. One of the few academic papers

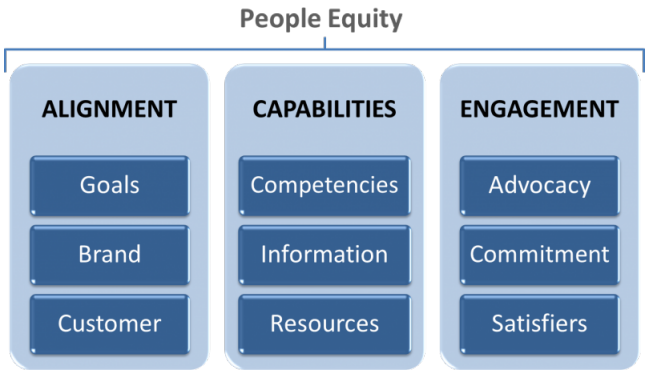

Figure 5: People Equity: The ACE Framework (Schiemann 2014)

touching upon the subject by Bamberger et al. (2006), shows some linkage between high-tech entrepreneurial startup firms growth and innovation, and the firm's willingness to compensate well in order to attract and retain talent, as well as creating a work environment which is structured loosely, in such a way that the attracted talent can work somewhat autonomously. Moser et al. (2015) study investigates the effect of employer branding and legitimacy on startup attractiveness and found strong support for employer branding as a predictor for startup attractiveness. The study found that the overall well-being of the employees had the highest effect, while the legitimacy of the startup founders had the lowest, albeit significant, effect on startup attractiveness.

In a guest lecture for the course "Technology-enabled blitzscaling" at Stanford University, the co-founder and CEO of Stripe, Patrick Collison, argues that it takes a long time to hire the best talent. One of the reasons is that the founder has to build legitimacy to make talent comfortable joining a startup. Collison (2021) continues by arguing that there are two ways that you can approach hiring the best people: either you filter by interest and then secondary filters for the good ones,

or you find the good ones and convert them to express interest, where the latter is the most effective according to Collison. This is backed by Spolsky (2007), who argues that the best software developers are never actually on the job market. The reason for this according to Spolsky is that great students are often identified during their studies and are hired prior to graduation or that they only apply to the few workplaces where they want to work, which emphasizes the importance of a strong employer brand.

2.2 Employer branding

One of the early contributions to and predecessors of the employer branding literature were from Gatewood et al. (1993) and Belt & Paolillo (1982) who explored the idea that applicants could be influenced by the image of a company which would make the company more attractive. A decade later, Cable & Turban (2001) published a paper that developed a concept about how job applicants were affected by their employer knowledge. The paper showed that the applicant's knowledge about the employer is valuable for the organization as it affects how the applicant process information about the employer. Furthermore, Cable & Turban (2001) demonstrated the different sources of employer knowledge and how the applicant is processing them. Finally, Lievens & Slaughter (2016) adds color to the research area by a thorough review. They find in the literature that there are several different terms used for similar constructs and suggest that an organization's external employer brand, which refers to someone outside the organization's mental image of the attributes connected to the organization as an employer, can be mapped with an organization brand image. Additionally, an organization's internal employer brand, which refers to the aforementioned example except with

someone inside the organization, can be mapped to an organization's identity. In Lievens & Slaughter (2016) review, two different perspectives are identified: The elementalistic perspective, which refers to instrumental and symbolic attributes, and the holistic perspective, which refers to the overall ratings of organizational attractiveness.

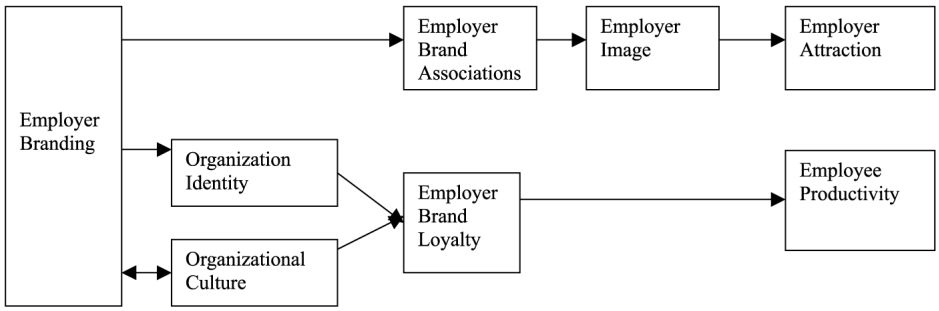

Figure 6: Example of An Employer Branding Framework by (Backhaus & Tikoo 2004)

2.2.1 The elementalistic perspective

The elementalistic perspective builds on the employer image as a sum of attributes that someone associates with the organization. In the recruitment literature, there has been a focus on symbolic and instrumental attributes. Lievens & Highhouse (2003) created the instrumental-symbolic framework for the recruitment field, wherein the case of recruitment, the instrumental attributes refers to applicants associations to pay, location, and benefits, i.e., more tangible attributes. It is important to note that the instrumental attributes are not general and therefore vary between the type of job and industry (Lievens & Highhouse 2003). In the case of symbolic attributes, it refers to attributes of an intangible and subjective nature, such as prestige. Highhouse et al. (2007) argues that these symbolic attributes are

important to employees and potential applicants since they use them to show that they are valuable and to impress their peers. Studies have shown that symbolic attributes tend to be more generalizable than instrumental attributes.

2.2.2 The holistic perspective

The holistic perspective complements the elementalistic perspective and Collins & Stevens (2002) argues that there are two elements of employer associations; perceived attributes and attitude, where attributes are defined as in the elementalistic perspective, and attitude refer to the general feeling that an applicant has to an organization. Even though parts of the holistic perspective remind of the elementalistic, there is a significant conceptual difference. The holistic perspective does not see employer image as the sum of attributes that someone associates with the organization. Instead, it emphasizes the overall attitude and feelings towards an organization. Collins & Kanar (2014) refers to this as surface associations and recruitment researchers have found that this surface-level organizational attraction is positively connected to the decision making in whether to take or decline a job. More importantly, it is also shown that surface-level organizational attraction has a higher positive effect on decision making than organizational attributes, e.g., elementalistic (Chapman et al. 2005).

2.2.3 Effects of employer branding

Much of the well-cited literature on employer branding is citing the classic marketing paper "Conceptualizing, measuring, and managing customer-based brand equity" by Keller (1993). In essence, the work by Keller discusses the incremental value of a strong brand, referred to as brand equity, that only exists because of the

companies name and the associations to it. There are obvious similarities to the holistic perspective from the employer branding literature. The positive effects of brand equity, such as decision making, differentiation, and loyalty can also be found in employer branding. For example, studies have shown that applicants more remembered recruitment material from companies with a stronger employer brand. Although in the marketing literature, it is shown that consumers are willing to pay a premium for strong consumer brands. Studies have shown that the same behavior is seen in recruitment, where people are willing to accept lower pay to work for an organization with a strong brand (Cable & Turban 2003). Turban & Cable (2003) 's study show that firms with a better employer brand attract more applicants, and some evidence suggests that these firms also attracted better talent. Chhabra & Sharma (2014) 's research also finds evidence of a significant positive correlation between brand image and the likelihood to apply.

2.3 Organization in startups

Since the dawn of humanity, there have been organizations in different forms. The object of an organization is to achieve goals, and in order to achieve these goals, they are broken down into multiple tasks that form specific roles. These roles can be grouped and form departments, and together departments form an organizational structure .(Daft 2020, Lunenburg 2012) Every organization has different goals and exists for different reasons, and therefore there are several different organizational structures. According to Mintzberg (1978), the characteristics of an organizational structure fall into natural clusters that he calls configurations. These configurations are built on several characteristics such as age, size, and the organization's industry. When the characteristics are mismatched, the organization does not fall into its

natural cluster and does not function efficiently. Mintzberg argues that there are six basic parts of the organization: strategic apex, technostructure, middle line, support staff, operating core, and ideology.

- In the operating core, the basic tasks are done, in other words, the production of products and services.
- The strategic apex is where the firm is managed and here we find the top management (i.e., C-suite executives).
- In the middle line, we find all managers in the middle between the operating core and the strategic apex.
- In the technostructure, we find the analyst who designs the systems, e.g., work processes and outputs of the operating core.
- The support staff includes the people who are not a part of the operating core but provides support, i.e., staff in the cafeteria or public relations.
- Lastly, the ideology is the beliefs and traditions surrounding the organization and makes it unique.

In the article "Organization design: fashion or fit?" by Mintzberg (1981) he describes that an organization is created by an entrepreneur that has an idea and forms the strategic apex of the organization. For the organization to be functional, people are hired to conduct the basic tasks that form the operating core. As the organization gets bigger, there is a need for people who work in the intersection of duties between the strategic apex and operating core, and therefore managers are hired to form the middle line. Lastly, some organizations need staff that plans and controls the work, which forms the technostructure and staff in the cafeteria and

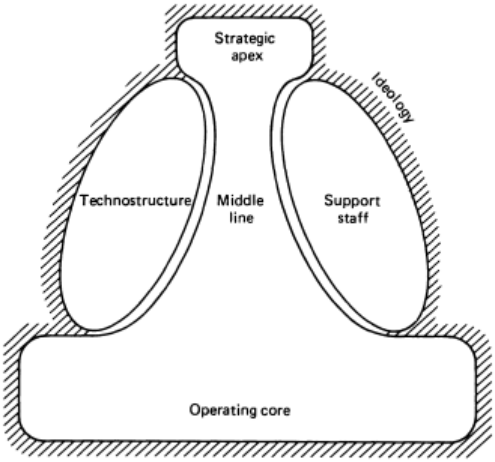

Figure 7: The six basic parts of an organization (Mintzberg 1978)

public relation, for example, that forms the support staff. According to Mintzberg (1981), all organizations do not need all parts of the basic organization mentioned above and are more simple. At the same time, some organizations combine all of the basic parts to form a complex organization.

2.3.1 Simple structure

The most common simple structure is the entrepreneurial company (Mintzberg 1981), and given that the companies in this thesis are startups, the simple structure will be in focus. In Mintzberg "The Structuring of Organizations" he describes the simple structure as one large unit, where there are one or a couple of top managers in the strategic apex and a few operators who perform the basic work. Additionally, Mintzberg argues that what characterizes the simple structure is what is missing from it. For example, as there is little focus on the standardization and formalization of the organization, such as training and planning, there is also little need for staff analysts. As mentioned earlier, the strategic apex handles much

of the management of the operating core, and for that reason, no or very few managers are hired for the middle line. The direct line between the operating core and the strategic apex is what creates the power of the simple structure (Mintzberg 1978). The simple structure is often found in dynamic and simple environments

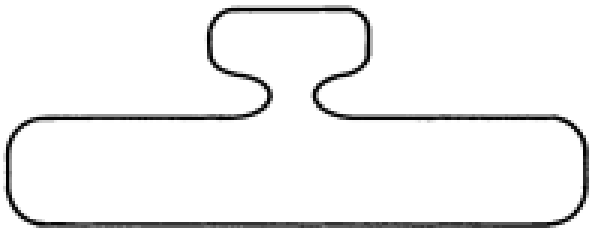

Figure 8: A simple organizational structure (Mintzberg 1978)

in order for the organization to be flexible so it can outmaneuver bureaucracies, and the simple environment and system of production, in order for the dominant one in the strategic apex to maintain centralized control (Mintzberg 1981) easily. As the organization ages and grows, the simple structure fades out, and the need for bureaucratization increases, which is why most organizations with a simple structure are small and recently established. However, many organizations with a simple structure never reach the state where the bureaucratization increases as they fail long before. One of the main reasons for failure is the dominant one in the strategic apex mismanaging the organization, which presents the main vulnerability of the simple structure where the power is centralized. As put by Burton et al. (1998), if the chief executive makes good decisions, the company will be successful, if not, it will fail. Furthermore, as the simple structure often lacks managers in the middle line, the whole organization's efficiency is limited to the strategic apex's ability to process information. This can be handled by specifying who in

the strategic apex has the responsibility of which task. If not, there is a risk of information overload in the strategic apex (Mintzberg 1978).

2.4 Software development in startups

In contrast to the well-established simple structure, there is not a clearly defined path in which successful software development is conducted in startups. Instead, as noted by Giardino et al. (2016), there is a general lack of studies on startups in general and software development in startups in particular. Most software startups are product-oriented in the initial phase of the development process, but as the business scales, managing the organization and software development becomes more complex, which can affect the overall software development performance (Lehman 1980, Hilmola et al. 2003). The structure of the software development process tends to grow over time, however, as there is little or no time for internal training as argued by Sutton (2000), there is an increasing focus on the firm's ability to hire software developers that do not require training and can add value from day one (Yoffie & Cusumano 1999). As described by Mintzberg (1981), the environment in newly started companies with a simple structure is often dynamic, whereas the flexibility and ability to react quickly characterizes the strengths of the simple structure becomes essential. Agile methods (Abrahamsson et al. 2017) is the response to this in the world of software development. It is suited for the dynamic and uncertain environment, using trial and error to learn from mistakes quickly and being able to work closely with the customer in order not to spend unnecessary time developing functionality that is unwanted (Midler & Silberzahn 2008, Sutton 2000). It is also common that software development practices known and used by the developers in the startup before are used, as it is the only tool

that they have, as noted by Coleman & O'Connor (2008). However, Coleman & O'Connor argues that good practice would be to invest time into collecting experience from other resources. Additionally, in the cases where there is a lack of engineering experience and best practices to learn from, would benefit from using external resources and follow software development practices that have been validated.

Lately, some focus of software development research has been on technical debt Giardino et al. (2016), and Tom et al. (2013) has presented five dimensions of technical debt: design and architecture, environment, knowledge distribution and documentation, code, and testing. The practices presented in the previous paragraph emphasize the development teams' ability to be agile and quickly adapt to the customer to reach product-market fit. A potential effect of this is the trade-off between speed and quality, from where technical debt could be created (Giardino et al. 2016).

2.5 Software bugs and issues

2.5.1 Detection

As digitalization increases and the world becomes more reliant on software, the importance of software quality increases. In order to ensure that the software has sufficient quality, engineers use several different methods such a testing and code reviews. Defects in the software, more commonly renowned as bugs, can result in an enormous cost for the company, and therefore much effort is invested in making sure that the software does not fail due to a bug (Briand et al. 1993, Johnson

et al. 2013). There are several ways of approaching bug analysis where to two most common ways are statically and dynamically. By approaching bug analysis dynamically, the bugs will be detected during the execution of the code. Tools which use the dynamic approach are more accurate but may affect the efficiency and run-time of the executed code (Li et al. 2006). By instead using a static approach to detecting bugs, the code can be analyzed without actually running the code. There are several ways to perform an automatic static analysis. It can be done continuously while writing code, at request, or when committed to external systems such as version control systems (Johnson et al. 2013). One of the key benefits of static analysis according to Li et al. (2006) is that there is not an overhead added upon execution, slowing down the system.

2.5.2 Analysis

There have been some studies looking into predicting the severity of bugs. The ability to predict the severity is critical, as it indicates how soon it has to be fixed, helping software development teams to prioritize (Lamkanfi et al. 2010). For example, a bug in a feature used by the masses might have higher priority than a bug in a feature only used by a fraction of the users. In software development teams, this is oft ben described as the severity of the bug. However, Lamkanfi et al. (2010) raises an interesting discussion on the topic, as the priority of a bug is relative to other bugs and other factors such as the next release, the severity becomes the only definitive classification. In an ideal world, each individual in the software team would consequently mark the severity of a bug the same, and most projects have clear guidelines in order to facilitate this. However, since humans are involved, it is not bulletproof. Lamkanfi et al. (2010) study investigates how

to prioritize bugs and finds that it partly depends on the severity, but in order to handle the human factor, they were able to predict the severity well with the help of other indicators, such as the textual description of the bug.

Efforts have also been put into analyzing how the programming language affects the software quality in terms of bugs. **?** uses projects from GitHub and analyzes how language features such as static, dynamic, strong, and weak typing affect the quality of the software. The study used over 700 projects and controlled for effects such as team size, project age, and size. It is shown that programming language has a significant effect on software quality, however, it is modest. Furthermore, project size, team size, and commit size are the dominant factor from which the modest effect arises, bringing little clarity to the subject.

2.6 Complexity analysis

Several methods and factors play a part when discussing and analyzing software quality and bugs, as shown in the previous sections. In software development, the complexity of a code is a common discussion, and several measurements are available to calculate this. One of the more well-known measures is Cyclomatic Complexity which was developed by McCabe (1976). Cyclomatic complexity is a graph-theoretic measure that measures the number of linearly independent paths in a section of a program's source code. For a path to be linearly independent, it has to have at least one edge that is not a part of any other path. The program's control flow graph is used to compute the cyclomatic complexity, which is a graph notation of all possible paths when a program is executed. Another well-known technique was introduced by Halstead (1977). His method uses several properties

of the code, such as the number of distinct operators, in order to calculate measures of the code, where one of the most popular measures of complexity is Halstead's Volume (V) (Zhang et al. 2007). Lastly, the oldest complexity metric is source lines of code (LOC), a size-based complexity measure that calculates the lines of code in software, excluding comments and blank lines. Using LOC as a complexity measure has been criticized due to the method's inability to capture the fundamental differences between programming languages. However, several studies are proving that LOC is highly correlated with the widely accepted cyclomatic complexity from McCabe (Graylin et al. 2009). Similar results were found by Herraiz et al. (2007) and Graves et al. (2000), and Herraiz et al. (2007) argues that with respect to how easy LOC is to compute, it is an interesting characteristic for the complexity of software.

3 Methodology

In this chapter, the methodology used in order to answer the research question is presented. A thorough explanation of the data collection, cleaning and analysis is also presented.

3.1 Research Design

This study aimed to investigate how founders and management should organize to make better technical decisions in a startup to enhance growth and generate less technical debt. The topic of this research has emerged from discussions and experience, both academical and practical. As mentioned earlier, there is a lack of research on this topic, making a qualitative approach appropriate as one of the primary benefits of the qualitative approach is to discover new variables and relationships (Shah & Corley 2006). However, there are elements of the technology decisions where a quantitative approach would be more appropriate. Therefore, this study applied a mixed-methodology approach, a combination of a qualitative and quantitative study. As mentioned by Shah & Corley (2006), a combination of qualitative and quantitative methodology creates a more rigorous theory. Additionally, combining the two would allow building an empirically grounded theoretical framework and then using the quantitative method to test and extend the theory. Due to the high complexity and under-research area, the theoretical framework was built on initial informal discussions with startup founders and investors but mainly through a thorough literature review. As the theoretical framework has been built, the quantitative data were analyzed, provided by the researched companies, and semi-structured interviews with decision-makers in the company, mainly CTO's,

CPO's, and founders.

The study used an abductive approach which is a combination of deductive and inductive approaches. Dubois & Gadde (2002) recommends the use of an abductive approach when the aim is to find something new, which suits the under-researched nature of this subject. The approach started with a pre-study in a deductive manner where a theoretical framework was built based on initial discussions with startup founders and investors and a thorough literature review on the topics of the thesis. This suits the advantages of a deductive approach as it helps explain relationships between concepts and variables (Saunders et al. 2015). As the theoretical foundation was built by the deductive approach, the study's inductive part started and further investigated and tested the identified relationships. The inductive approach started with observations from the theoretical foundation, pre-study, and discussions with industry experts. In the inductive part of the study, an exploratory multiple-case design was used as it enabled to investigate the similarities and differences between cases in order to provide a more robust result (Yin 2017, Gustafsson 2017). Additionally, the investigated case companies were startups in the initial phase of their journey, meaning that the number of employees was rather few, limiting the number of interviewee objects within a single case company. Therefore, a multiple-case study was sufficient in order to generate enough data by interviewing employees of several different startups. Furthermore, as argued by both Yin (2017) and Gustafsson (2017), this allowed me to analyze data within each case company and across different companies. Concerning the researched area in general and technical decisions such as programming language in particular, this serves as an advantage since a single case company most likely only chose one or very few programming languages, which would limit the impact of this

thesis. The interviews were analyzed by coding as proposed by Cope (2010), in order to find patterns which theory could be built upon. Furthermore, the bug data was analyzed quantitatively, guided by the observations from the inductive approach.

3.2 Data collection

Proposed by both Yin (2017) and Blomkvist & Hallin (2015) the case study approach elevates as the use of sources increases. As stated earlier, this study's angle on the research area is highly complex and is of an under-researched nature, adding to the importance of using several sources. A multiple-case study has been conducted in this study where the primary data sources have been gathered in different formats. In total, 15 companies have participated, where four have both been interviewed and shared quantitative data, six have only been interviewed, and five only shared quantitative data. The aforementioned interviews and quantitative data acted as the primary sources of this study. The secondary sources have been gathered through an extensive literature review described below, informal discussions with experts, and surveys from well-renowned publishers.

3.2.1 Literature review

In order to get an overview of the researched areas, a thorough review of the current literature was conducted. As the thesis touches upon several different research areas, the literature review was divided into four main topics: Talent Management, Employer Branding, Organizational theory in startups, and bug analysis. The purpose of the literature review was to build a solid foundation on the various topics which the research touches upon, get an overview of the state of the current

literature and its knowledge of the researched area, and acted as a springboard to interesting areas of research related to the research question.

The literature review was based on several different forms of media such as books, research articles, peer-review journals, industry reports, company reports, and statistical reports. As the literature touches upon practitioner-oriented subjects such as software engineering and talent management, and especially in the context of startups since the environment that they operate in develops rapidly, there is a potential risk for shortcomings in the literature review by only focusing on formal literature such as peer-review papers and journals (Garousi et al. 2019). Therefore, grey literature has been included to enable the identification of emerging research and thoughts from several different voices and angles. Grey literature is defined by Schöpfel & Farace (2010) as:

> *"grey literature is produced on all levels of government, academics, business and industry in print and electronic formats, but which is not controlled by commercial publishers, i.e., where publishing is not the primary activity of the producing body"*

In the world of startups and venture capital, there is a significant amount essays and blog posts written based on observations and experience. Schöpfel & Farace (2010) puts emphasis on the importance of source selection in grey literature and provides a framework for this which has been applied when using grey literature in the review. As the research covers several different topics, the literature review was divided into four individual reviews. A summary of the types of literature and keywords used per topic is provided below.

The review process itself was performed in the same manner for all research topics. The approach I took was first to use the available online search engines to scan and quickly oversee the material for each topic. The approach to finding the best literature was made by searching in layers. Firstly, Google Scholar was used as an initial filter due to its superior relevancy in its search function. The result from the Google Scholar search was then filtered and analyzed, and in order to validate the quality of the filtered search results, Web of Science and Scopus was used, which ensured a degree of quality since both mostly contain peer-reviewed journals. Secondly, the filtered and quality certified literature was subject to a deeper analysis where key themes were identified for each topic and divided the literature review as such (e.g., see division in 2.1). The identified key themes were then reviewed individually, and the current state of the literature for each theme was summarized. The identified key themes also had to fill the criteria of being highly relevant for constructing the theoretical framework to help answer the research question.

Topic	Literature type	Keywords
Talent Management	8 books, 18 peer-reviewed journal articles, 1 company reports, 1 Stanford lecture	Recruiting, talent management, attract, retain, develop, talent
Employer Branding	14 peer-review journal articles	Employer branding, employee branding, employer value proposition, brand equity
Organization in startups	3 books, 1 peer-review journal, 1 research article	Mintzberg, simple structure, organization, startups.
Software development	1 book, 7 conference papers, 12 peer-review journal articles, 1 research article	software development, startups, bug analysis, complexity analysis, software complexity, bug detection

Table 1: Overview of literature review by topic

3.2.2 Interviews

As the deductive part of the study came to an end, the inductive part started with a series of semi-structured interviews with CTO's, founders, and fund-side CTO's (i.e., employees of the venture capital funds that has a technical background and often takes a more hands-role in new portfolio companies). There were several motivations for the semi-structured interviews, firstly to enable data triangulation in order to increase the reliability and validity of the study (Shah & Corley 2006, Easterby-Smith & Thorpe 2002). Additionally, the semi-structured interviews served their purpose by giving an understanding of the complex decision-making and the reasoning behind it from several different viewpoints. The semi-structured interviews allowed to use open-ended questions. Furthermore, it enabled the flexibility to follow the interview path and, based on the responses, direct the conversation, which serves the exploratory approach of the case study and the research questions.

The interviews were between 30-60 minutes long and were held with decision-makers in the startups, mainly technical background and responsibility such as the Chief Technical Officer (CTO) and the Chief Product Officer (CPO). The reasoning behind the chosen interviewees is that they have the best insights into the decision-making as they most likely took the decisions and the most knowledge and expertise in technical decision making.

The interviews took place both face-to-face and via video conferencing due to geographical reasons and the constraint brought by the ongoing pandemic. As the interviews were conducted alone, taking efficient notes while driving the conver-

sation was an apparent challenge. Therefore, all interviews were recorded after consent from the interviewee was given.

#	Industry	Position	Interview length	Funding stage	Employees
1	Healthcare	CTO	48 min	Post-seed	6
2	Supply chain	CTO & Co-founder	45 min	Post-seed	7
3	Sales automation	CTO & Co-founder	50 min	Pre-seed	12
4	Property technology	CTO & Co-founder	30 min	Seed	8
5	Education technology	CTO	35 min	Post-seed	40
6	Venture capital	CTO & Partner	31 min	N/A	8
7	Retail technology	Engineering Specialist	44 min	Seed	5
8	Marketing technology	CTO	47 min	Post-seed	23
9	Learning platform	CPO	45 min	Post-seed	4 (10+*)
10	AI learning platform	CTO	47 min	Seed	15

Table 2: Interview overview. *Company 9 have scaled down for the moment due to Covid-19, the interview questions were answered based on the pre-covid environment.

3.2.3 Issue data

The issue data was collected from the companies with two different methods. In some cases, short-term access to their GitHub repositories was given, which enabled the use of GitHub's REST API and cURL queries to export all their data to JSON format. If access was not given to their repository, cURL commands were provided to the companies, which they could use to download the GitHub issues by themselves. However, a Python script that used the REST API to export and pre-filter all data into a CSV file was also developed, which saved significant time since the data cleaning described below was not necessary in those cases. In some cases, participating companies used software other than GitHub, such as

Jira, Trello, and Notion. In those cases, the export was more straightforward and provided by the company.

3.2.4 Data cleaning

The issue data retrieved from the companies' GitHub repositories are in a JSON format which is a simple and common way to store data. Each JSON file from the companies was merged and unnecessary data such as who created the task were removed in order to make the large amount of data easier to handle. Furthermore, a command that filtered out all data entries that were not labeled as a bug was used, and this is important since many companies also use GitHub issues for other purposes. The same cleaning procedure was used when the data were retrieved from other sources. When working with issue trackers, it is relatively common that some issues are forgotten or never resolved due to various reasons. Those cases did not help in answering the research question. Therefore the dataset was cleaned by excluding all issues with more than 45 days of completion time, which was calculated by looking at the distribution of completion times.

3.3 Data Analysis

3.3.1 Interviews

The interviews were transcribed by listening to the audio recordings and manually transcribing the conversation. As this stage is in the inductive part of the study, the data analysis of the interviews was done simultaneously as the data collection described in previous stages, and the interviews were analyzed thematically (Saunders et al. 2015). The thematic analysis was done in three stages: Comprehending,

synthesizing, and theorizing. The comprehending phase was done along with the data collection and consists of a general coding of the collected data based on a first reading that was performed to get an overview of the data and identify and understand the main themes of the interviews. In the synthesizing, the text was read once again and recurring patterns were identified. The identified patterns were grouped into themes that represented the specific patterns and keywords. The themes were mostly picked based on the theoretical framework. However, decision-making was chosen based on practical relevance for the thesis. The theorizing phase, followed by the synthesizing phase started, where the theoretical concepts were developed by identifying and defining the relationship between the coded data generated from the previous steps. This was followed by testing the generated theoretical concepts

3.3.2 Issue data

The quantitative issue data retrieved from GitHub were analyzed by aggregating the data based on different attributes such as programming language, financing stage, number of employees, and direct reports. At first, the number of issues labeled as a bug per programming language was calculated to get an overview of how the distribution looks. Then, the number of issues was normalized with respect to the total source lines of code. The reason for normalizing is that in some languages like C and C++, where the developer has to handle memory, such errors are common but also easy to handle, whereas the number of bugs might not be the most efficient metric to answer the research question. This was followed by looking into how long the average time between opening and closing such an issue took. In a startup environment, much functionality is due in a short time span as

customers start using the platform. Therefore, the metric of how long it takes to complete issues, with respect to the total source lines of code, acted as a more interesting measure.

3.4 Quality of research and ethics

The methodological choices of this thesis have been made based on careful considerations and argumentation, with the aim to reach the highest standards of research quality and, therefore, reliable results. For example, the study used a mixed-methodology approach that, according to Shah & Corley (2006) creates a more rigorous theory. Furthermore, the multiple-case study approach reaps the benefits of a strong focus on generalizability, validity, and rigor Yin (2017), while also mitigating the effects of concluding a small number of samples, which may harm the generalizability of the results Gustafsson (2017). Additionally, concerning the fast-paced and under-researched environment that startups operate in augurs a high likelihood for contrasting results, in which a multiple case approach can be beneficial Yin (2017). However, the thesis was written under a time constraint of four months. The multiple case study approach could result in a lack of depth in the understanding of the subject compared to a single case study approach, as argued by Dyer Jr & Wilkins (1991). As expressed by Gerring (2004), the more cases a study has, the confidence in its likelihood to be representative increases, but the time spent observing each case also decreases. Furthermore, the study complied with ethical standards by ensuring anonymity for all participants as individuals, but also the companies that they represented. The reasoning behind this was to enable the interviewees to speak more freely without the risk of sharing industry and company-specific secrets.

4 Results

In this chapter, the results from the themes from the interviews and issue data is presented. The identified main themes were: Organization, decision-process, software engineering, and talent management. Each theme has sub-themes that will be presented under each main theme.

4.1 Software Development

Four sub-themes were identified in the parts of the interviews focused on software development in startups and related decisions. The first sub-theme was scooping which refers to scooping features and projects. Secondly, software development method was identified, which refers to the working method such as agile methodologies. Lastly, two related sub-themes were identified: speed and quality, which refers to focus and sought-after effect when implementing and building software. A general finding from the interviews related to software engineering is that there is a strong focus on the speed of implementation in the initial phase of the startup rather than the quality. However, those of the interviewed startups that have passed the initial phase and are maturing, highlighted that the focus had shifted more to quality (Interviews 5, 8, 9).

> *"I would say we've transitioned, when we built our first version, it was speed. So we made trade-offs and we aggregated technical depth and designed it. Now we're focusing much more on quality and have turned down larger clients that make requests that would require too much of a technical refactoring or design refactoring."* - Interview 9

On the other hand, the CTO's in interviews 1 and 4 argued that choosing a pro-

gramming language that they are comfortable with enabled them to emphasize both the quality and the speed of the implementation. However, the CTO in interview 1 emphasized quality, while the CTO in interview 4 focused more on speed.

4.1.1 Scooping

As pointed out by the CTO's in interviews 1 and 4, there can be methods that arguably enable both focus on speed and quality. In several of the interviews, one of these approaches was scooping. In the context of the interviews, scooping refers to the process of taking a feature and identifying its very core functionality. Although the scooping process was done slightly differently in each of the startups that applied this methodology, in company 3 they looked at what parts of the feature would generate the most user happiness.

> *"So that first we look at the whole, like the goal of the feature, how would the dream scenario look with this feature? And then we try to scale it down to its very core and develop that thing with quality. So we're like peeling off the outer parts and then just develop a small part, which is going to generate huge, like user happiness effects."* - Interview 3

While in company 8, they had a two-faced approach by first only building features that are useful for all users. Secondly, scaling down the requirements according to the CTO made it much easier to fix and with increased user happiness. An interesting finding is that in both interviews 3 and 8, the CTO's pointed out that the scaled-down feature meets all the customer needs.

> *"Tech-Wise we scaled down the requirements, which made it much*

> *easier and turns out that the thing we've fixed seems like it covers most of the use case for that feature. So I think it was a very good decision. It increased the value of a product with a relatively relatively small efforts."* - Interview 8

Additionally, another way of scooping, according to the CTO of interview 8 is only to build functionality that can be used by all customers. This is a common approach for SaaS platforms as its value lies in its ability to scale easily.

> *"So if a customer has a request and if we wanted to make that into a feature, it has to be something that everyone can use. So we can't build customer-specific things. So that has been the most important thing. If we could not fit it into that, then it would not be built."* - Interview 8

However, apart from only identifying the core of a feature or more extensive implementation, some of the interviewees talked about the scope of a feature from a different perspective. For example, in interview 2 the CTO argued that they focus more on the scope of the feature within their current architecture. Suppose the feature is something that they will not continue to develop, or more importantly, build additional features upon or that relies on it. In that case, they will focus more on the speed of implementation and more or less neglect the technical debt created from the implementation, with the motivation that the created technical debt does not significantly affect the rest of the software. However, if it is a feature that is a core part of their software that several other features will rely on, then quality is the main focus point. On the other hand, the CTO from interview 2 and several other CTO's acknowledges that there is a balance between speed and quality that has to be considered on a case-by-case basis regardless of the identification of a

core or non-core feature.

> *"I would say that at the higher level, it's a balance between how many things do we have that needs to be taken care of versus like new development. Like if this pile with the quick and dirty features is just growing and growing, eventually that's your platform."* - Interview 2

The CTO of Interview 5 has a similar method the CTO of interview 3; instead of only focusing on if a functionality is a core part of the system, they also weigh in on how sure they are that the feature will live for a long time. In other words, based on the collected customer data, how convinced are they that this feature will be used and therefore considered as a core part of the system. The CTO of interview 5 explained it as if a feature is "experimental" and requested by a small portion of their customers and they decide to implement it, the focus will only be on speed, and quality will be neglected. The CPO from interview 9 added some color to this way of working. He argued that as long as something works for the user, they will not know if the code base is a total mess, or as put by the CPO: "spaghetti code".

4.1.2 Speed & Quality

As mentioned at the beginning of this section, there is a clear focus on speed in the initial phase from almost all interviewees, a focus that is shifting as the company and software it is building becomes more mature. However, as pointed out briefly in the previous section on scooping, there are situations where there is always a focus on speed or quality, no matter how mature the software is. For example, a majority of the interviewees argued that decisions regarding the architecture of the

software should always have a strong emphasis on quality. The common theme of that argument was that if you make the decisions regarding the architecture right, it will help develop other parts fast. Another common argument was that since the environment that a startup operates in is very dynamic, and the amount of data in terms of requests and feedback from customers is extensive, it requires an architecture that allows for quick changes in order to handle the fluctuating environment.

> *"You base your choices on the information you get from the first customers, but then it might come other customers that work in a different way, and then you have to reevaluate those choices. And hopefully you have an architecture that can facilitate those changes."*
> - Interview 1

However, another common theme in the discussion revolving around speed or quality is how technical debt is created. A clear majority of the interviewees argue that no matter how much time is invested in making a good decision and focus on quality, technical debt will always be created, especially when creating something new. Related to this, many of the interviewees mentioned that they build the software very fast to get feedback fast and then learn from the effects of the decisions taken. One example of this is from interview 5, where the CTO stated that they are in the initial phase with few customers and are building features that they know will break sooner or later, and then they learn and redo with an iterative process. In interview 9, the CPO stated that when building the minimum viable product (MVP), which is often in the initial phase, then you should not focus too much on building the perfect architecture. Instead, it would be best if the focus was on getting the main thing working. Then when it is time to build

the first real version, use the learnings from building the MVP and invest much time in getting a good architecture. Regarding this, it was mentioned by some interviewees that the "build fast, learn fast" strategy has its limitations and is constrained to the characteristics of the customer group. For example, in interview 9, the CTO mentions that their primary customer is people working in HR, which he characterized as a slow-moving group resulting in longer feedback loops. Based on the other interviews, this phenomenon seems attributable to most business-to-business (B2B) scenarios. In contrast, the CTO in interview 9 argues that if the product were business-to-consumer (B2C), the feedback loops would be shorter and even more suitable for a "build fast, learn fast" strategy.

4.2 Programming languages and issues

Apart from the interviews, a quantitative study analyzing the relationship between programming language and time spent on fixing issues was done. In total, 571 628 lines of code were analyzed, with a total of 1423 reported issues in six different languages. From the interviews, it was found that the choice of technical stack was somewhat similar, resulting in an uneven spread on issues between the languages, as shown in the figure below. As mentioned earlier, looking at the number of issues per language might not paint an accurate picture. Therefore, by looking at the number of issues compared with the number of issues normalized with LOC, we can see which language produces the most issues per line of code. In figure 9 below, we see that the least amount of issues per line of code was produced with C# .Net and Python. However, the number of issues reported in C# .Net and Python is far fewer than in other languages. TypeScript produced the most issues, but it produced the third least number of issues on a normalized scale. Thus, together

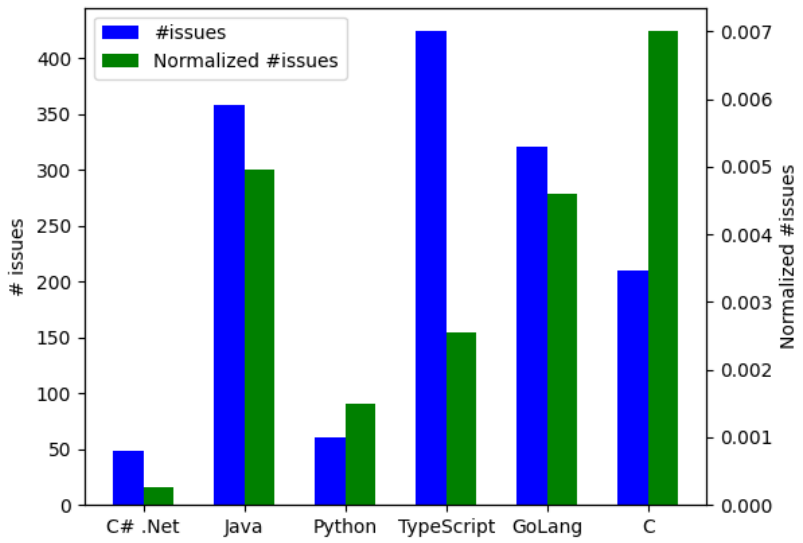

Figure 9: The amount of issues per programming language

with C# .Net, we find that TypeScript has the most significant difference between the number of issues and the normalized number of issues. On the other hand, we find that Python and C produce more issues on a normalized scale.

To get a better overview, we look at the average time spent to fix an issue, also normalized on the LOC. The results here are similar to the number of issues. Both Python and C have a longer fix time on a normalized scale, indicating a greater time spent fixing issues per line of code. As with the number of issues, we find that C# .Net and TypeScript have the most significant difference between the absolute average fix time and the normalized average fix time. Additionally, they also have the lowest average fix time on a normalized scale. We also find that in Java and GoLang, the difference between average spent fix time and normalized average fix time is close to zero, indicating that the time spent scales linearly with the

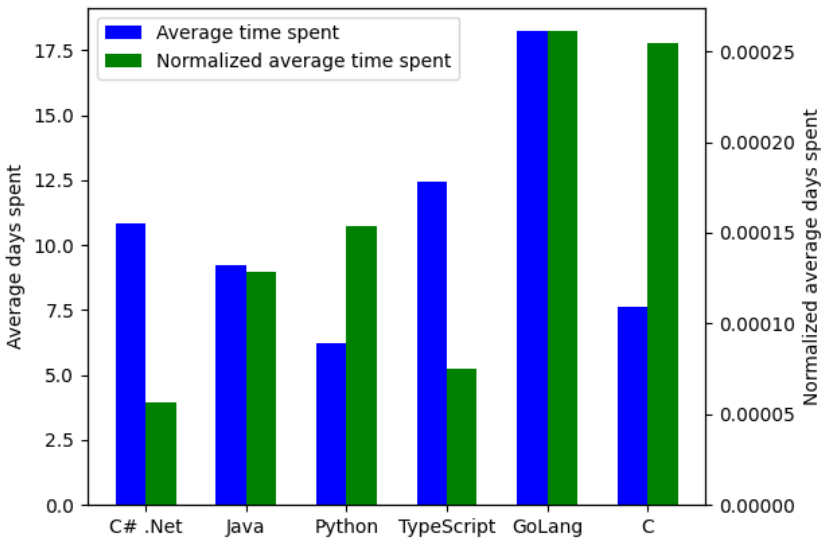

Figure 10: The average time spent in days on fixing issues per programming language

LOC.

4.3 Talent Management

In the interviews, there were many motivations to decisions with its basis in question regarding talent management. As presented in the introduction, there are many people knowledgeable in a few programming languages (Stack Overflow 2020), and a general finding from the interviews is that CTO's were very aware of this, and it showcased clearly in their decisions presented in the following section. In the interviews regarding talent management, the main identified sub-themes were access to talent, signaling, and employer value proposition. Access to talent refers to discussions from the interviews related to the size and quality of the

talent pool and how technical decisions were related to this. Signaling refers to discussions revolving around how certain technical decisions made the company more attractive to work at. Signaling is a part of the employer value proposition, but it was discussed to such an extent that it will be presented as a standalone section. Lastly, we have the employer value proposition, which refers to discussions related to the section in the theoretical background.

4.3.1 Access to Talent

Two of the interviewed companies had chosen to outsource parts of their development team to outside of Sweden. One of the companies had outsourced to Poland while the other to Ukraine. One interesting finding when discussing the talent pool was that it seems to differ depending on the country. The CTO from interview 3 mentioned that the talent pool in Poland had a completely different skill set compared to a similar talent pool in Sweden. For example, in Poland, the major language and framework were C# .Net, while very few used JavaScript frameworks. The CTO mentioned that the talent pool's attitude toward technical stacks was also different compared to the one in Sweden.

> *"Actually we had some negative effects by choosing node.js and TypeScript, when they realize we are using node.js as backend. Polish developers they view node.js as like vanilla JavaScript, which is a not a very attractive language to work with."* - Interview 3

In interview 5, the CTO had outsourced their development to Ukraine. When choosing to develop their software with Python, he had done extensive research on the talent pool in Ukraine and knew that Python was not the most popular language.

However, he knew that there were many skilled developers and argued that Python is easy to learn and, therefore, it did not matter too much. Furthermore, he admitted that it has become harder to attract python developers. In contrast, it has been effortless to find developers to work with their frontend, which is built with Flutter, a new and superhot framework. Although not stated clearly by all interviewers, a similar pattern could be found in other cases where the choice of language in the frontend was more motivated by the access to talent or acted as compensation for a smaller talent pool in the chosen backend language.

> *"Well, I would not say entirely, but a large part of it was because of that [access to talent]. Because like you said, it is one of the most popular frameworks right now and you cannot go wrong by picking react basically. And I knew react. I was comfortable with it, but then also there are so many people that knows it. So that was definitely true. And with Golang, I would say it's probably hurt me more than anything in that regard, like if I wanted access to all the talent in Nordics specifically."* - Interview 2

On the topic of local talent pools, the CTO in interview 5 stated that it is also essential to look at what the big companies in the local ecosystem uses, as it could be hard to compete with them. Additionally, several CTOs argued that the programming languages used by big companies were often the ones that consultancy firms and boot camps (similar to yrkeshögskola) trained their employees or students in.

> *"You get everybody on those technologies and not only people that are actually passionate, not only people that that love what they do, but*

you get everybody. You get some people that are passionate as well, but you also get the people that are choosing it just because they want to have a place of work." - Interview 5

According to many of the interviewees, the result of choosing a programming language used by the big industry companies was a decrease in the quality of the talent pool. Moreover, some also stated that the talent pool became less passionate about programming, as many had learned the programming language due to its high demand in the job market.

"But there's a lot of people these days switching careers and stuff. They did something else and now they want to do coding. So you have a lot of these quick educations or boot camps." - Interview 9

4.3.2 Signalling

In the interviews, many of the CTO's mentioned that it was hard to find talent, especially as the big tech companies such as Spotify and Klarna swiped the market and could often offer better salaries and perks. However, a common way to mitigate this was to use the latest technology and frameworks in order to attract talent. Several of the interviewees mentioned that they experienced potential employees to be more interested when they mentioned that they worked with the latest technologies.

"Google Cloud, Kubernetes, microservices, event-driven design, all of those are just buzzwords. And every time you say a new buzzword word, you know, it just sparks in the eyes of the interviewee." - Interview 3

On the other hand, two CTO's who had tried using signaling also mentioned that

there were risks. For example, the CTO in interview 9 chose to develop their backend in Haskell, which according to many, is considered to be a language that has a high signaling value. As a result, they effortlessly attracted good talent but later found that the talent was not there for their passion for the problem that the company was solving. Instead, they were there for their passion for programming in Haskell and therefore focused on matters that were not important for the company.

> *"As I said, they [Haskell developers] were very bad at communicating with non-technical people. Even the frontend developers had a tough time understanding the Haskell developers, which is a very big problem in our tech team ... So a feature could take longer time to build customer arguing about which case to use for variables, for example. So there was a lot of like opinions which was very hard to manage."*

Building on a similar point, the CTO in interview 10 pointed out that talent attracted to the latest technology tends to be harder to retain. According to the CTO, if the company does not update to the latest technology continuously, then there is an overhanging risk that the talent will look for new opportunities for the same reason that they joined in the first place. Even though also arguing that it could be an excellent way to attract talent, he referred to it as "temporary attractiveness for a year or two".

4.3.3 Employer Value Proposition

Many interviewees mentioned that it was hard to attract technical talent in general, especially since the big tech companies in their local ecosystem were exceptionally

good at sweeping the market for talent. However, many worked actively with building brand equity and their "employer value proposition". For example, we found that most of the interview CTO's had thought of how they can stand out among other startups in general and the big tech companies in particular. The most common approach was to communicate how the working life at a startup differs from the life at a big tech company. Examples of such communication from the interviews were that in a startup, the employees would be able to see the impact of their work as they are a more significant part of the organization. As put by the CTO in interview 9, the developer will build, test, and get feedback from the end-user. The CTO from interview 3 used a similar approach but had more focus on how the work-life environment looked at their candidate's former or current employer (big engineering companies). They communicated that at their company, they would be able to grow outside of the "developer bubble" and grow fast within the organization as they are in the scale-up phase.

> *"Often they come from enterprise companies, they are tired of the huge processes of being a small piece in the system. So we try to focus on that you will create your own microservice every month, that you are a massive part of it. You have contact with customers if you want to, because some developers do not want that, but you can like grow out of the developer bubble. We're also building the core team now, so you will be able to be a manager and grow fast within the company."* - Interview 3

Similar points were made from several CTO's. For example, the salary was mentioned as a factor where startups had a hard time competing with the big tech companies. One CTO mentioned that he understood that the salary was important,

but he also argued that good culture, career path, growth possibilities, ownership, and the feeling of having an impact and working together towards a tangible goal are more important than salary.

The CTO from interview 10 took another approach to attract people. Apart from using the differences in the work-life environment mentioned above, they also communicated a clear mission for their company in general and their product in particular. This was a key factor in their employer value proposition that helped them recruit talent.

4.4 Managing a startup

In all of the interviews, two sub-themes of the organizational structure were identified related to technical decision making, and those were knowledge and communication. The following explanations of the sub-themes are based on the context in which they were used in the interviews. Knowledge refers to how the competence and general knowledge within the organization are distributed among employees and how that affects the organization's ability to make decisions, and if the decision-making is centralized or decentralized. Communication refers to how the organization's structure (or lack of) affects how components of the organization are communicating and how information is flowing through and within the organization. A finding from the interviews related to organizational structures, in general, is that only one company had intentionally organized in a specific way, i.e., having a thought-through organizational structure. At the same time, the rest argued that it was a product of unintentional trial and error.

"I don't think we have placed too much emphasis on the organizational

structure yet. We have the classical with the CEO at the top, I guess going down to me (CTO) and then from there, everything involved with the technical product development. Then we have two in sales directly reporting to the CEO" - Interview 2

In the interviews, it was found that all companies had a simple organizational structure where the founders and management were in the strategic apex, and all non-management and founders were in the operating core, having their areas of responsibility. However, as startups grew in terms of headcount, the strategic apex became more divisionalized based on function. Apart from divisionalizing the strategic apex, several interviewees mentioned that team size was an essential factor in deciding their organizational structure.

"We've tried out like a lot of different structures, it is difficult. I would probably organize it a lot differently if it would be 20 people." - Interview 3

In all companies, we found certain functions, as either a department or one person with that responsibility, and those were sales, product, and technology. Some of the investigated startups had additional functions, but what differentiated the companies was the interplay between the functions. In interview 5, the CTO states that even though they have departments, there have not been any formal structures, and the reasoning behind this is that this is the only way to be truly agile when working and making decisions. However, he acknowledges that they would need formal structures as they grow more, and the reason it has worked until now is that everyone in the management team has over 15 years of startup experience and knows exactly what to do and expect from each other.

"It has been a very conscious decision to work this way. And one of the reasons why is that, at least in my experience, this is the only way to truly be able to be agile. I mean, being a startup, we have had a great need for agility and what I mean is to be able to extremely quickly adapt to customer and market needs, often within weeks or even days. And what you actually lose by that is the ability for long-term planning" - Interview 5

4.4.1 Knowledge sharing

Several of the interviewed startups stated that a significant portion of the knowledge in the company was centralized to the founding team. For example, in interview 2 the CTO stated that almost all product-related decisions required the input of the CEO (founder) since he was the only one in the organization with prior experience in the domain that they operated in.

"So a lot of things kind of falls back on him, but not because he's the CEO, but more because he has the knowledge basically." - Interview 2

In contrast, in interview 1 the CTO explained that no one in the organization had specific domain knowledge of the area they operated in, resulting in more decentralized knowledge in the organization and internal discussions in order to spread knowledge.

"We are trying to keep it pretty open and discussing in the team because we all have different experiences and knowledge" - Interview 1

The actors in the organization had different backgrounds and experiences, and therefore they tried to involve as many as possible in order to share knowledge by

discussing. In interview 7, the CTO mentioned that they have people of different backgrounds, but in contrast to interview 1, they have employees with domain knowledge in most of the teams.

> *"So we don't have direct contact with them most of us, but we have a background experience from their [the customers] field."* - Interview 7

According to several interviewees, the result of decentralized knowledge is that there can be a lack of understanding between the functions. The lack of understanding between functions created misalignment within the company. Some of the mentioned effects were prioritization issues (i.e., important tasks not being done in favor of completing unprioritized tasks), functionality not meeting customer requirements, and time invested into realigning teams. In the interviews, it was identified that this problem was most common between the development and customer-facing teams. A common way to mitigate the communication issue between the development and customer-facing teams was through changing the way actors communicate within the organization, which will be brought up in the following section about communication.

> *"Misunderstandings between people, people tend to be in the same way if you work within the department. So there can be a miscommunication between departments"*- Interview 8

The CTO from interview 10 explained that they were a very mission-driven startup and that their product had a clear purpose. According to him, this made it much easier to get everyone on the same page and that the knowledge spread within the company became less important since the mission drove a lot of thought and decisions.

> *We are a very sort of mission-driven company and it's very easy to describe the purpose. I mean you get the job to have people who are employed at a low level to be able to take on more challenging roles, to become a team lead, usually, you grow people. Which means it's very easy for us to sort of get everyone on board on why we're doing things. I think that we're enough to sell that product.* - Interview 10

In the interview with the CTO and Partner of a Venture Capital fund, he added some color to this area from both an investor perspective as well as a CTO perspective from prior experiences as a CTO at a startup. He argued that there are not many ways to organize in the beginning when there are only a few employees, including the founding team.

> *"In the early stage there's usually only maybe two people or three people, and then they're not too many different ways to organize sort of formal sense."* - Interview 6

However, without regard to how the formal organization structure looks, he argued that all involved should have decision-mandate in their expert domains to maximize their time. In other words, that there should be as few middlemen as possible in the process of getting something done, and with respect to the scarce resources in startups, too much overhead in terms of transparency and communication might hamper the team.

> *"In general making sure that all people involved have sort of the power to do things on their own, it's about maximizing the use of their time. And so usually a good sort of division of responsibilities and the authority to make decisions regarding each person sort of own domain*

with as few sort of middlemen as possible" - Interview 6

4.4.2 Communication

In order to mitigate the effects mentioned above of centralized and decentralized knowledge, the interviewed companies often took organizational measures to improve the communication between team members and functions. For example, in interview 8 the CTO stated that they had included the development team in the discussions with customers to mitigate the risk of making wrong decisions as a result of miscommunication. The CTO of interview 7 also did this. However, he also mentioned that by including the development team in the customer meetings, he had experienced that the meeting resulted in more of a technical support Q&A session. In contrast to the other interview startups, the CTO of interview 1 stated that he was actually the one closest to their potential customers.

> *"Developers are also active in customer discussions when it is needed. So it is correct that the customer success and sales are mostly the ones mostly communicating with customers, but they also bring in development. People want us to know!"* - Interview 8

The CTO of interview 3 did not mention that developers were attending meetings with customers and instead focused on the information loss when channeling information to through different teams.

> *"I guess one challenge is to channel all the feedback from the sales, marketing customer success department into product, and then into a tech, like to keep those perfectly aligned, because there is always information like disappearing on the way, that is a problem, I guess*

for all organizations if you are not just like one big giant team, but then you would not be able to coordinate anything." - Interview 3

The most common way to mitigate communication and knowledge issues within organizations was to restructure the organization and set up clear information flows. For example, the CTO of interview 3 stated that they have iterated on several structures and noticed that the communication was not good. Therefore, they added a product team in the organization's center that would own the communication between sales & marketing and the development team. By doing so, he argued that the development team only received the information that was relevant for them in order to do the implementation.

"Previously, we did not have a product team. We had a design team and then we had the two other teams and that was it. Then we experienced problems and we converted the design team to like put more product-focused team which was great because it was less operative and more, it became more strategic kind of... Now the information from sales and marketing goes into the product team and then into the development team" - Interview 3

The CTO of interview 5 used a similar method but instead let the customer success team handle the communication between the customer and the development team. However, instead of only acting as a bridge between the customer and the development team, the customer success team also handled the prioritization of tasks for the development team. The CTO described their workflow as follows and later added that by moving prioritization away from the development team, they could solely focus on the implementation and therefore making the responsibilities

clear.

> *"For feature requests and customer pain points, we document all the customer requests that flow in. And we have a process where the customer success team prioritizes. They do the prioritization of all customer requests. So they do a more or less a 1-5 prioritization and the requests that are of the highest priority, then we have an ambition that we include at least two of from that top 10 list. We include at least two requests in each release."* - Interview 5

Both company 3 and 5 has reached a phase where the organization is divided on function, and therefore the installation of customer success or product team is more natural. However, in interview 2 it was found that the same issue with communication had been approached similarly while still being in the initial phase of the startup. Since company 2 only has 5 employees, they created an "imaginary" product team consisting of the CEO and the CTO as they did not have a dedicated employee focusing on the product. The imaginary product team acted as the bridge between the customer conversation and the development team was created.

> *"I would say like the salespeople and the CEO would have the customer requirements, which is then put forth to like the imaginary product team, which would be me (CTO) and the CEO"* - Interview 2

The CPO in Interview 9 had a similar setup as the one presented by the CTO in interview 3 above and described the most prominent problem as "bridging sales and product development". However, apart from setting up the specific team structure, they took it one step further and tried to align sales and product by having regular demos throughout the development process. By doing so, the CPO argued that the

product development team could get feedback in several phases to ensure that the final version of a feature was precisely what everyone expected.

> *"Product developments main job obviously is listening to the users and clients to understand their needs. But it's also internal needs or internal insights or ideas that come from both marketing and sales that can be very valuable. So it [the regular demo sessions] gave us a chance to incorporate insights from them into the process."* - Interview 9

In contrast to the rest, the CTO in interview 4 had another methodology to handle the centralized or decentralized knowledge. They assigned one in the development team to be responsible for gathering information regarding a specific technical decision and testing a simple solution. In the world of software development, this is referred to as a "Spike". There are two different types of spikes, technical and functional, and in interview 4 they used technical spikes with the aim to familiarize the team with new technology and its impact on the current implementation (Abrahamsson et al. 2017). This was then presented to the development team and all stakeholders, followed by a group decision.

> *"Whenever we come in front of a decision where we are going to select a technology we usually do spikes." - Interview 4*

In interview 6 with the CTO and Partner at the Venture Capital firm, he suggests a similar way of thinking based on his own experience from startups. In terms of information flow, he argues that the CEO should only communicate the strategy and the product roadmap and then leave it to the technical team to decide on and implement the technology. However, in contrast to the majority of founders in

terms of communication and transparency, he argued that it is more important to be dynamic in the approach than making sure that all parts of the organization are up to date with the latest information. The reasoning behind this was that it is better to remake or change the approach based on customer feedback than to have a well-organized way of guessing the best approach.

> *"I would say it is more about being flexible and able to change your focus or the way you are going and less about being fully transparent and communicative in all directions."* - Interview 6

4.5 Decision Making

The theme revolving decision making was in most cases related to one of the aforementioned themes, such as ensuring that the correct information is accessible in order to make a better decision as in the communication section, or taking a decision based on a focus on speed or quality as in the software development section. Therefore, this section will focus on how decisions were motivated and how the decision domains looked in the interviewed companies.

The most common decision mentioned during the interview was the choice of technical stack. This decision was motivated by several factors in each company. However, some factors occurred in the majority of the interviews. One surprising finding from the parts of the interviews focusing on decision making was that the clear majority of CTO's did not choose their technical stack and, more particular, their programming languages based on what they had most experience or knowledge in. Out of all interviews, only three based their choice mainly on

what they knew, while the rest had other more important factors. For example, as mentioned in the talent management section, more than half of the interviewees motivated their choice of programming languages partly on the access of talent and how that choice would make them more attractive as an employer, a part of the employee value proposition described in the section about talent management. Apart from access to talent, there were several who combined that with looking at the popularity cycle of programming language in order see what language that is gaining or losing popularity. There are many changes in popularity, so it is important to look closely at how the trends look. As put by the CTO in interview 1, when building something that people will pay for, it is a good idea to chose a mainstream stack that has been around and will be around for the next couple of years.

> *"My background is 10 years of Python. So going here and doing Golang, it's a bit weird actually. But the other developer had more experience with Golang. The next step from Python is going to Golang if you look at the popularity cycle in the world. So I was kind of keen to learn it. So the decision is more or less that, and from experience hiring developers in Stockholm is super hard, super complicated. And if you're not Klarna or Spotify you need to have at least something else. And then new popular programming language could be a good thing for hiring people and not staying too far back." - Interview 7*

Another clear motivator in the technology stack decision, used by almost all interviewees, was the simplicity of the programming language. The attribute of a programming language being simple was often related to enabling rapid and easy development, easy to collaborate in as a team, easy to learn, and the talent pool

was big and skilled so hired employees could hit the ground running.

> *The main drivers for the choices were that it would be a stack that allows for rapid development which, of course, narrows things down quite quickly, and that it allows for relatively simple way of finding skilled developers. - Interview 5*

A factor in technical decision-making that was mentioned by surprisingly few interviewees was the needs of the software. Only two interviewees mentioned this as a factor and showcased two different sides of it. For example, in interview 9, the CTO argued that when he made the big technical decisions in the company's initial phase, he thoroughly investigated the needs of the software. One example of such a question that he asked himself was if the speed of the software was important for the customers. He argued that answering those questions would significantly help him in motivating and making the necessary technical decisions. In interview 10, we saw a similar approach. However, instead of looking at what was necessary for the customers, they used user data and demographics to motivate technical decisions. One example of such as decision was to create a web-based application instead of a mobile application since, for the majority of their customers, downloading an app would be a critical obstacle.

5 Discussion

This chapter contains the analysis of the empirical data presented in the previous chapter with a starting ground in the theoretical framework presented in chapter two..

5.1 Organization

As presented in the background, newly started companies often fall under the simple structure presented by Mintzberg (1981), and the companies interviewed during the data collection of this thesis were not an exception. All of the companies used a simple structure, however, it was often divided into smaller divisions and the reasoning for this according to the majority of CTO's was to make it easier to organize and make the responsibilities clearer. Furthermore, some of the CTO's argued that the reasoning behind keeping the organizational structure as simple as possible was to meet the demands of the customers, who often have a high expectancy on the delivery pace of features. This reasoning is aligned with one of the benefits of the simple structure according to Mintzberg (1981). Mintzberg argued that the simple structure is often found in dynamic environments, where the simple structure has an advantage compared to bureaucracies since it can be more agile.

What makes the simple structure agile is its centralized power in the strategic apex (Mintzberg 1978). This means that there are very few actors and layers in the decision-making process, enabling the decision-making process to be quick and agile. However, this also results in a lack of formal structures and the organization gets exposed to the quality of the decisions in the few people in the strategic apex.

As mentioned by Burton et al. (1998), if the strategic apex makes terrible decisions, the company will fail. In the interviews, it was found that information and knowledge often were centralized to the founding team, i.e., the strategic apex. As put by one CTO, their founder (CEO) had to be present in almost all discussions revolving around decisions since he had all the product and domain knowledge. As argued by Mintzberg (1981), there is an overhanging risk of information overload when the responsibilities within the strategic apex are unclear. What was found in this case is that there is an overhanging risk of information overload when the knowledge of the domain is centralized to one or very few in the strategic apex. A potential solution for this is to invest time into spreading the domain knowledge in the strategic apex, which could be done by including more parts of the organization in the customer conversation. Another interesting finding, although only mentioned in one case, is that a mission-driven startup could potentially reduce the harm of centralized knowledge. A well-defined mission could help in finding the correct decision, regardless of the domain, since the aim of the decision is to fulfill the mission. In the interview with the Partner and CTO of a VC fund, he argued that one should decide over their own area of responsibility. However, if the knowledge is centralized, then it would not be possible to divide the responsibility. For example, a CTO without domain knowledge cannot take many technical decisions themselves since many technical decisions related to the software have a strategic impact and therefore often requires domain knowledge. There were also cases where the knowledge was decentralized, i.e., no one with significantly higher domain knowledge than others. In those cases, an open discussion was often required which could slow down the decision process. Furthermore, it was found that there was a lack of understanding between functions which created a misalignment. If this is only

due to the decentralized knowledge is unclear, however, one could argue that the informality of open discussions presents an opportunity for misinterpretation.

The most common way to handle the aforementioned issues was to create channels of communications and set up specific teams. It was found that most problems were between the customer-focused teams and the development team, where CTO's experienced that the gap between them was to big. In the interviews, several CTO's argued that creating a product team, which from an organizational standpoint is in between the customer and the development team, that created a bridge where information and knowledge were processed from the customer to the development team. In companies with fewer employees, the team product team was imaginary, represented in the form of a weekly meeting for example. From Mintzberg (1981) it is found that creating more divisions based on function more resembles a bureaucracy, which most certainly does not suit the environment and needs of a growing startup. However, the created product team did not operate as regular teams. Instead of having an operational purpose in the operating core, the team had more of a strategic purpose, belonging in the strategic apex. Centralized or decentralized, this results in less stress on the individuals in the strategic apex, but more importantly, a way to spread and aggregate knowledge in both scenarios.

5.2 Software Development

When looking at making decisions that enhance growth and creates less technical debt, the speed and quality of the implementation are interesting factors. As argued by Lehman (1980), Hilmola et al. (2003) and seen in the interviews, startups are very product-oriented in the initial phase of the development process. It was found

in the interviews that this often plays out by working closely with customers from where feature requests or issues are gathered. Moreover, as mentioned earlier, the environment is fast-paced and dynamic and the result is a lot of pressure on the development team to deliver within a short time frame. To handle this with high quality is challenging, as argued by Giardino et al. (2016), and the interviews showed that there were several different approaches.

The most common approach was to scope the future to its core and then to build that with quality. It was argued by many CTO's that the scaled-down features still fulfilled the customer needs. This way of approaching software development is in line with Coleman & O'Connor (2008) who argues that it is a good practice to invest time and do research. By researching the feature, the companies were able to scale it down, implement it quickly and with quality. Even though this method seems to work very well, it still might take some time to do the initial research. To handle this, another strategy was presented, which consist of looking at the architecture of the software and the requested feature and judge how that feature will scale. In other words, if it is something that other features will rely on, then it will be built with quality. Otherwise, it will be built with speed. In this case, it is crucial to consider the reasoning of one CTO who argued that if you build stuff "quick and dirty", that will eventually be your whole platform. By taking this balance between speed and quality into consideration, it is found that scooping seems to be the better approach since scaling the feature down does not necessarily generate technical debt, while it enables quick development and based on the interviews still satisfied customers. However, this stresses the importance of the discussions in the previous section on knowledge and communication. To successfully scope features to their core requires extensive knowledge of what

the customers want. Scaling down features that are important for the customer might significantly harm customer satisfaction and, in the worst case, result in a churn.

From the interviews, it was clear that the focus was on speed in the company's initial phase where the MVP is built. However, it was also clear that as the business matured, the focus shifted to quality. This is expected behavior since as the company finds product-market fit, it becomes less dependent on testing new features. One CTO had an interesting argument regarding how to approach the development of the MVP. It was argued that when building the MVP, it is worth disregarding technical debt in favor of speed. This enables the company to test features and get quick feedback to speed up the process to reach product-market-fit. From which, the software is rebuilt based on the learnings from the MVP and with a focus on quality. A similar trade-off between speed and quality has been argued by Giardino et al. (2016). By instead focusing on quality in the initial phase would not be a great choice for many reasons. Firstly, even if the founder and team had great experience and knowledge, they would most likely not be representative of their whole market. Building software requires a general solution that suits many, and therefore it is essential to get as much feedback as quickly as possible. Thus, neglecting speed in favor of quality would not be an efficient approach to reach product-market-fit.

5.3 Talent Management

Attracting technical talent is an extremely hot topic and something most companies struggle with, small and large. However, it is not easy for startups to compete

with big tech companies on a financial level in terms of attracting talent with salary, which results in an even harder challenge for startups. Furthermore, an important difference between established companies the interviewed companies is that the latter all had a simple structure. This is important when discussing recruitment as it is a process, which is a challenge for the simple structure. With a talent pool that is already small and with the fact that only three programming languages were known by less more than 50% of the talent pool in mind, the effect the technical choice has becomes interesting (Stack Overflow 2020). Even with the apparent urgency to work with talent management, the amount of CTO's who actively work with and consider access to talent in their technical decision-making was surprising given the small number of discussions regarding this in the literature and popular media. In the literature, Smart (1999) argued that every employee of the organization should be an "A-player", Michaels et al. (2001) while argues that employees should be divided into performance categories, and lastly Collings & Mellahi (2009) argues for a differentiation based on the strategic impact of the role. An important aspect is that these arguments have their basis in an established company. A startup is very different. Due to the small number of employees, every position has a significant impact and therefore has a strategic impact. This means that the importance of hiring the right candidate becomes much more important in a startup than in a big company as the relative effect is higher. In the interviews, it was found that to handle this with respect to the small talent pool. Two companies chose to outsource their development, most likely due to financial reasons since the size and quality of the talent pool are similar. However, it was found that the skill-set within the talent pool differs based on geographical location, i.e., that the programming languages and technical frameworks known by the majority vastly

differed between countries. This is not so surprising when considering the rapid pace of development and shifting trends in the developer community. Furthermore, how far the country has come on the path to digitalization might also be a factor. Outsourcing or not, the fact that technical decisions have an effect on access to talent is clear and confirmed by all interviewees and actively used as a factor in technical decisions by the majority.

As the competition for developers increases, some CTO's tried to work with employer branding in order to get ahead of their competition. One example of this is signaling. Several of the CTO's mentioned that they experienced an increased interest from candidates when they branded or talked about their modern technology choices. This behavior can be explained by what Collins & Kanar (2014) calls surface-level associations and is a part of the holistic perspective (Collins & Stevens 2002). Recruitment-researches have shown that surface-level attraction is positively connected to decision-making whether a candidate will accept a job offer or not (Chapman et al. 2005). This suggests that image can be an important tool for startups to compete with the big tech companies in the war for talent. However, evidence of indirect negative effects was also found. Although not directly related to the signaling and brand image in particular, one CTO mentioned that by attracting talent with the help of technologies as a "signal", they attracted exceptional talent. However, that talent was only there because of the ability to work with that particular technology and did not have passion for the mission or fitted the culture, resulting in a net-negative effect since much time was needed to be invested in order to manage them. For example, the CTO mentioned that the hired developers focused on technicalities that did not matter for the company, but were important for their "ego". Furthermore, one CTO argued that talent who

joins because they want to work with the latest talent is harder to retain since they will more likely get tired of the technical stack and seek new opportunities. In the literature on retaining talent, a low score on the People Equity framework measures is linked with low retention (Schiemann 2014), and the talent in the aforementioned situation would score very low since it is more aligned with the technology than the company.

Another approach to stand out from the big tech companies was to work with the employee value proposition, which was coined by Michaels et al. (2001). As several studies conclude that the salary is not the most important factor in attracting talent (Hiltrop 1999, Michaels et al. 2001, Zaharee et al. 2018), an opportunity for startups to compete arises since the more important factors of attraction such as being challenged and the opportunity to grow within the organization have a more leveled playing field. In the interviews, it was found that many startups already work with this when they attract talent. Instead of focusing on the salary, they instead push on the impact the employee could have, growth opportunities, and challenges. According to the CTO's, this is a very effective approach since a majority of the talent comes from big companies with a completely different employee value proposition, and this gives startups an opportunity to stand out. In the same way, there were successful cases of attracting talent by branding the companies mission and how employees could contribute to fulfilling the mission.

It is clear that working with an employee value proposition is a great way for startups to attract, retain and develop talent. However, building a strong brand image takes time and requires resources, both financial and human. On the other hand, there is a lot of low-hanging fruit, such as defining the startups' mission or communicating how the startup life differs from the life at big companies.

Additionally, with respect to the literature that shows that a better employer brand attracts more talent and that people were willing to accept lower pay to work for organizations with strong employer brand (Cable & Turban 2003, Turban & Cable 2003, Chhabra & Sharma 2014), it becomes more clear that it is a good strategy. However, the use of signaling should be used since it attracts talent, but with caution and in combination with checking the cultural fit.

5.4 Decision Making

There were several different factors in motivating technical decisions, and most of them are related to situations in the sections above. The most common technical decision was the choice of technical stack, and as mentioned in the talent management section, many decisions were motivated by access to talent. A clear majority did not motivate their choice based on their own knowledge, instead they looked at what attributes would be important for the customers. Additionally, there was a strong focus on choosing a programming language that was easy and enabled quick development. This is aligned with the literature where Yoffie & Cusumano (1999) argues that there is an increased focus on hiring developers that can hit the ground running, and choosing an easy language could potentially help in that. Surprisingly few of the CTO's looked into the needs for their software, one explanation for this might be that most languages are capable of doing almost everything. However, it is also important to consider that some languages are better in some areas. If the software requires niche attributes, then this would be an important factor. In the interviewed companies, most have developed somewhat standard web applications which might explain the low focus on this factor. The only CTO's who mentioned the needs of the software as a factor in their decision-making process were the ones

with more complex requirements.

From the results, it was found that there are several different methodologies to ensure that the right information is at the right place at the right time. However, the structure of decision-making does not seem to be a focus in the interviewed companies. Some of the CTO's mention how they make decisions, and the most common way is that the CTO makes the decision with the CEO based on the results of a discussion. As discussed earlier, a lot of time was invested in creating a better information flow in order to share knowledge in the organization. This creates an opportunity to structure the decision-making process to suit the new information flow, creating a more decentralized decision-making process that does not require the presence of the CEO as the input from the CEO is already integrated into the information flow. Such a structure on the decision-making process could potentially also put less pressure on the strategic apex that often already have a high amount of information flow, and decentralizing the process also mitigates the risk of being vulnerable to the quality of the decisions of one or a few individuals in the strategic apex, concerns which were raised by Mintzberg (1981).

5.5 Effects of Programming Language on Value Creating Activities

As mentioned earlier, the most commonly discussed technical decision was the choice of programming language. Based on the interviews, it is clear that the ease of development was an essential factor when choosing a programming language and the stack in general. In the quantitative study on issues, it is clear that there is a significant difference between different programming languages when looking

at the time spent fixing issues. Interestingly, the number of issues (figure 9) does not seem to depend on whether the language is statically or dynamically typed, as found by (Ray et al. 2014) in their study looking at code quality. It is important to note that Python was the only dynamically typed language in the study and created the second least number of issues per line of code. Furthermore, the total number of issues in the best performing languages (C# .Net and Python) were also the ones with the least amount of reported issues, which potentially raise concerns regarding the validity of the data in those specific languages. Looking at the bigger picture, TypeScript had the most reported issues while having the third least issues per line of code, which gives indications that TypeScript generates fewer issues in relation to the total lines of code. Additionally, the TypeScript data comes from several different projects, making the results more likely to be representative on a general level. However, as mentioned in the theoretical framework and by Herraiz et al. (2007) and Graves et al. (2000), even though LOC is highly correlated with widely accepted complexity measures such as McCabe's Cyclomatic Complexity, LOC's potential inability to capture the fundamental differences between programming languages might affect the results. Especially in the programming languages where the amount of data is lower. As mentioned earlier, only looking at the number of issues does not paint the whole picture when looking at the effects of programming languages. As in the case of the number of issues, both C# .Net and TypeScript had the greatest difference between the average time and the normalized average time spent on fixing issues. Once again, there does not seem to be a clear pattern indicating differences in performance between statically and dynamically typed languages. However, as both C# .Net, although with very few data points, and TypeScript performs well in both metrics, it provides some indications that there

could be a difference between programming languages in general. With regards to time spent on fixing bugs, dynamically typed Python performs badly as the time spent increases when normalizing on the total lines of code. This increase could be the result of more extensive testing, which particularly important with dynamically typed languages. However, it is also important to factor in that even though the data was cleaned, the general practice with working with issue tracking differs between projects. One example of this is the priority of an issue as discussed by Lamkanfi et al. (2010). Some of the investigated languages were only based on one project, and there there is a risk of not being representative.

6 Conclusion

In this master thesis, it has been investigated how startup founders and management can organize to make better technical decisions in order to enhance growth and create less technical debt. As the research questions were broad and covered many areas, it was divided into three sub-research questions, focusing on the factors of technical decision making, the effects of different programming languages on speed and quality, and lastly, how technical decisions affect the ability to recruit and retain talent.

In the literature, no studies looked into the effects of technical decisions or how to organize to make better technical decisions. Thus, this thesis is the first sod for researchers, laying the foundation in a yet under-researched area. From the practitioner's point of view, this thesis provides guidelines and points of interest for an internal revision of the decision-making process and highlights the important factors in technical decision-making and the subsequent effects.

6.1 Factors of technical decision making

When making decisions, there is always intangible psychological factors that arise from, for example, previous experiences. However, the findings show there are factors in technical decision-making that are found across all companies. The most prominent factor was, rather naturally, the ease of development. In other words, how the technical decision would affect the overall complexity of the software. Additionally, it was found that that this factor was two-fold depending on the stage of the startup. In the early stage, prior to finding product-market-fit, ease of development implied a focus on how the decision would affect the speed of

development to iterate on the product quickly and find product-market fit. While in the later stages, the focus shifted to ease of building with and maintaining high quality. Furthermore, it was found that access to talent also was a significant factor in the technical decision-making process, although not a deciding factor. Lastly, it was also found that companies with more complex solutions (ex AI, ML, statistical platforms) weighed software requirements more in their decision-making than companies with lower complexity (ex, simple web-based software) who had more focus on customer requirements.

It is also noted that the ease of development factor had a dynamic factor present on top of it based on how the current software architecture looked. This was referred to as having a balance between implementations that are built with the aim of being quick or with quality. In other words, the current quality of the software architecture is an essential factor when making a technical decision.

6.2 Programming language and time spent on issues

The findings show no clear evidence that the number of issues and the time spent on fixing issues are related to the language being either dynamically or statically typed on a broader level. This is in line with previous research where other intangible factors, such as personal preference, are considered more important. However, we find that C# .Net and TypeScript perform substantially better than other languages in both scenarios, and by also considering the number of data points, there seem to be indications that TypeScript has the best overall performance. This indicates that there could be a difference between programming languages in general. Additionally, since the TypeScript data comes from several different

projects, it is less likely that the results have been influenced by other factors than the performance of the language. However, when drawing conclusions on this data, it is of utmost importance to consider that companies work differently with issue tracking, and some of the investigated languages only come from one company. Furthermore, this study does not take the overall quality and seniority of the development team into consideration, which could potentially have an effect on the results.

6.3 Technical decisions and the attraction and retention of talent

As mentioned by several of the interviewed CTO's, finding technical talent is a difficult task, and what this study finds is that the technical decisions in general, and the choice of programming language in particular, substantially affects a companies ability to attract talent. However, conclusive evidence on whether it affects the ability to retain talent has not been found. More specifically, it was found that choosing a trendy programming language increased the attraction, but on the other hand, there was also some evidence that this decreased the ability to retain talent, as the attraction was temporary since it follows the trend. However, this is not conclusive. A critical aspect of choosing a trendy programming language related to retaining talent is that the attraction might be more to the language itself rather than the mission of the company, and some evidence is found that this often results in a cultural miss-fit and leaving talent, which is also supported in the literature (Schiemann 2014). It is also noted that the skills in the talent pools differ radically between countries, suggesting that "signaling" decisions and technical decisions in general, need to be curated on a local level.

On a more general note, the study finds that working with the employer value proposition of startups is a great way to attract talent. The technical decision is a part of the employer value proposition. However, we find that the strongest factors of the employee value proposition on the attraction of talent are the factors where startups significantly differ from other companies, i.e., individual impact (ability to make decisions, build for and see effects on customers), growth opportunities, and being challenged.

6.4 Concluding words

In summary, the study finds that a key factor in a startup's ability to make better technical decisions is to have the right information at the right time. It is found that the most effective way to ensure this is to create a product team that bridges the gap between the customer-facing teams and the development team. In very early-stage startups, this team can be imaginary, represented by a weekly meeting. Furthermore, this enables startups to decentralize their decision-making process, making them more fast-paced and less dependent on the knowledge of the strategic apex, mitigating the risks both mentioned in the literature and by the CTO's. Additionally, the improved information flow enables the startup to scope features, as the understanding of the customer needs is spread across the organization. It is found that scooping features to its core enables quick development while creating less technical debt, and at the same time generating high customer satisfaction. However, with a simple organizational structure, processes are a challenge. It highlights a need for a balance to still reap the benefits of the simple structure.

6.5 Sustainability implications

Sustainable development is often divided into three dimensions: economic, social, and environmental (KTH 2021). The purpose of this thesis was to get a better understanding of technical decision-making in startups in order to make better decisions. Better decisions most likely result in more successful startups that directly contribute to economic sustainability as it generates wealth for the founders and employees of the startup. Furthermore, successful startups will likely indirectly contribute to social and environmental sustainability, as radical innovations often emerge from startups, such as Tesla's electric vehicle. Additionally, this thesis has social sustainability impacts as it uses an ethical methodology approach with the use of anonymous and aggregated data to ensure that personal data is not compromised. Furthermore, all research on startups and how to improve the success rate of business ventures has both a social and economic impact on under-developed countries as startups do not require a developed business network to thrive. The emergence of successful startups in under-developed countries, which will improve the economy and generate jobs, benefiting society as a whole. Lastly, this thesis investigated how to organize in order to make better technical decisions in startups. Improved knowledge in this area has implications for social sustainability. It can create more effective organizations that are better suited for the challenges companies are facing today.

6.6 Limitations of research and suggestions for future research

In this thesis, there are some limitations that need to be considered. Firstly, the issue data from the participating companies have all been analyzed with exactly the same

method, and the data has therefore been regarded as the same. However, companies work differently with issue tracking and this is not shown in the data and this should be regarded when drawing conclusions based on the data. Secondly, this study applies a multi-case study approach, and this results in rather generalizing results. However, as the time frame of the thesis is small and the researched companies are many, the time spent investigating each company is small. This could result in a lack of depth in the study.

As a consequence of the limitations of the study, future research should try to increase the size of the issue dataset to represent more languages with more data points. Furthermore, as there is a potential lack of depth, future research should use the results of this thesis and redo it with a single case study or multiple case study with fewer cases in order to find more depth in the conclusions. In the setting of a single case study or a multiple case study with fewer cases, there is also a greater opportunity to control the issue dataset in order to ensure that the effects of different working methods are not visible in the data. Lastly, signaling is shown to be an effective way to attract talent, while it is also exposed to the current trend and therefore the temporary attraction. Future research within the field of computer sciences should look into how to predict the longevity of a trending programming language in order to enable a more calculated signaling choice.

References

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2017), 'Agile software development methods: Review and analysis', *arXiv preprint arXiv:1709.08439* .

Andreessen, M. (2011), 'Why software is eating the world'.
URL: *https://a16z.com/2011/08/20/why-software-is-eating-the-world/*

Ashton, C. & Morton, L. (2005), 'Managing talent for competitive advantage: Taking a systemic approach to talent management', *Strategic Hr Review* **4**, 28–31.

Backhaus, K. & Tikoo, S. (2004), 'Conceptualizing and researching employer branding', *Career development international* .

Bamberger, P., Bacharach, S. & Dyer, L. (2006), 'Human resources management and organizational effectiveness: High technology entrepreneurial startup firms in israel', *Human Resource Management* **28**, 349 – 366.

Becker, B. E. & Huselid, M. A. (2006), 'Strategic human resources management: Where do we go from here?', *Journal of Management* **32**(6), 898–925.
URL: *https://doi.org/10.1177/0149206306293668*

Belt, J. A. & Paolillo, J. G. (1982), 'The influence of corporate image and specificity of candidate qualifications on response to recruitment advertisement', *Journal of Management* **8**(1), 105–112.

Blomkvist, P. & Hallin, A. (2015), 'Method for engineering students. sl: Studentlitteratur'.

Boudreau, J. W. & Ramstad, P. M. (2005), 'Talentship and the evolution of human resource management: From professional practices to strategic talent decision science', *Human Resource Planning Journal* **28**(2), 17–26.

Boudreau, J. W. & Ramstad, P. M. (2007), *Beyond HR: The new science of human capital*, Harvard Business Press.

Briand, L. C., Thomas, W. M. & Hetmanski, C. J. (1993), Modeling and managing risk early in software development, *in* 'Proceedings of 1993 15th International Conference on Software Engineering', pp. 55–65.

Burton, R. M., Obel, B., Hunter, S., Søndergaard, M. & Døjbak, D. (1998), *Strategic organizational diagnosis and design: Developing theory for application*, Springer Science & Business Media.

Cable, D. M. & Turban, D. B. (2001), Establishing the dimensions, sources and value of job seekers' employer knowledge during recruitment, *in* 'Research in personnel and human resources management', Emerald Group Publishing Limited.

Cable, D. M. & Turban, D. B. (2003), 'The value of organizational reputation in the recruitment context: A brand-equity perspective', *Journal of Applied Social Psychology* **33**(11), 2244–2266.

Cappelli, P. (1999), 'The new deal at work: Managing the market-driven workforce'.

Cappelli, P. (2013), 'Attracting and retaining the right talent'.

Cascio, W. & Boudreau, J. (2010), *Investing in people: Financial impact of human resource initiatives*, Ft Press.

Chapman, D., Uggerslev, K., Carroll, S., Piasentin, K. & Jones, D. (2005), 'Applicant attraction to organizations and job choice: A meta-analytic review of the correlates of recruiting outcomes.', *The Journal of applied psychology* **90**, 928–44.

Chhabra, N. L. & Sharma, S. (2014), 'Employer branding: strategy for improving employer attractiveness', *International Journal of Organizational Analysis* .

Cohn, J., Khurana, R. & Reeves, L. (2005), 'Growing talent as if your business depended on it', *Harvard business review* **83**, 62–70, 155.

Coleman, G. & O'Connor, R. V. (2008), 'An investigation into software development process formation in software start-ups', *Journal of Enterprise Information Management* .

Collings, D. G., Scullion, H. & Vaiman, V. (2015), 'Talent management: Progress and prospects', *Human Resource Management Review* **25**(3), 233–235.
URL: *https://www.sciencedirect.com/science/article/pii/S1053482215000236*

Collings, D. & Mellahi, K. (2009), 'Strategic talent management: A review and research agenda', *Human Resource Management Review* **19**, 304–313.

Collins, C. J. & Kanar, A. M. (2014), 'Employer brand equity and recruitment research', *The Oxford handbook of recruitment* pp. 284–297.

Collins, C. J. & Stevens, C. K. (2002), 'The relationship between early recruitment-related activities and the application decisions of new labor-market entrants:

a brand equity approach to recruitment.', *Journal of applied psychology* **87**(6), 1121.

Collison, P. (2021), 'Blitzscaling 11: Patrick collison on hiring at stripe and the role of a product-focused ceo'.

Cope, M. (2010), 'Coding qualitative data', *Qualitative Research Methods in Human Geography* pp. 223–233.

Daft, R. L. (2020), *Organization theory & design*, Cengage learning.

Dries, N. (2013), 'The psychology of talent management: A review and research agenda', *Human Resource Management Review* **23**(4), 272–285.
URL: *https://www.sciencedirect.com/science/article/pii/S1053482213000296*

Dubois, A. & Gadde, L.-E. (2002), 'Systematic combining: An abductive approach to case research', *Journal of Business Research* **55**, 553–560.

Dyer Jr, W. G. & Wilkins, A. L. (1991), 'Better stories, not better constructs, to generate better theory: A rejoinder to eisenhardt', *Academy of management review* **16**(3), 613–619.

Easterby-Smith, M. T. & Thorpe, R. (2002), 'R. and lowe, a.(2002)', *Management research: An introduction* **2**, 342.

European Commission (2014), 'What is an sme?'. (Online, accessed 2020-01-31).

Fombrun, C. J. & Wally, S. (1989), 'Structuring small firms for rapid growth', *Journal of Business Venturing* **4**(2), 107–122.
URL: *https://www.sciencedirect.com/science/article/pii/0883902689900256*

Gallardo-Gallardo, E., Nijs, S., Dries, N. & Gallo, P. (2015), 'Towards an understanding of talent management as a phenomenon-driven field using bibliometric and content analysis', *Human Resource Management Review* **25**(3), 264–279.
URL: *https://www.sciencedirect.com/science/article/pii/S1053482215000212*

Garger, E. M. (1999), 'Holding on to high performers: A strategic approach to retention', *Compensation and benefits Management* **15**, 10–17.

Garousi, V., Felderer, M. & Mäntylä, M. V. (2019), 'Guidelines for including grey literature and conducting multivocal literature reviews in software engineering', *Information and Software Technology* **106**, 101–121.
URL: *https://www.sciencedirect.com/science/article/pii/S0950584918301939*

Gatewood, R., Gowan, M. & Lautenschlager, G. (1993), 'Corporate image, recruitment image, and initial job choice decisions', *The Academy of Management Journal* **36**, 414–427.

Gerring, J. (2004), 'What is a case study and what is it good for?', *The American Political Science Review* **98**(2), 341–354.
URL: *http://www.jstor.org/stable/4145316*

Giardino, C., Paternoster, N., Unterkalmsteiner, M., Gorschek, T. & Abrahamsson, P. (2016), 'Software development in startup companies: The greenfield startup model', *IEEE Transactions on Software Engineering* **42**(6), 585–604.

Graves, T., Karr, A., Marron, J. & Siy, H. (2000), 'Predicting fault incidence using software change history', *IEEE Transactions on Software Engineering* **26**(7), 653–661.

Graylin, J., Hale, J. E., Smith, R. K., David, H., Kraft, N. A., Charles, W. et al. (2009), 'Cyclomatic complexity and lines of code: empirical evidence of a stable linear relationship', *Journal of Software Engineering and Applications* **2**(03), 137.

Gustafsson, J. (2017), 'Single case studies vs. multiple case studies: A comparative study'.

Halstead, M. H. (1977), 'Elements of software science'.

Herraiz, I., Gonzalez-Barahona, J. M. & Robles, G. (2007), Towards a theoretical model for software growth, *in* 'Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)', pp. 21–21.

Highhouse, S., Thornbury, E. E. & Little, I. S. (2007), 'Social-identity functions of attraction to organizations', *Organizational Behavior and Human Decision Processes* **103**(1), 134–146.

Hilmola, O.-P., Helo, P. & Ojala, L. (2003), 'The value of product development lead time in software startup', *System Dynamics Review* **19**(1), 75–82.

Hiltrop, J.-M. (1999), 'The quest for the best: human resource practices to attract and retain talent', *European Management Journal* **17**(4), 422–430.
URL: *https://www.sciencedirect.com/science/article/pii/S0263237399000225*

Johnson, B., Song, Y., Murphy-Hill, E. & Bowdidge, R. (2013), Why don't software developers use static analysis tools to find bugs?, *in* '2013 35th International Conference on Software Engineering (ICSE)', pp. 672–681.

Keller, K. L. (1993), 'Conceptualizing, measuring, and managing customer-based brand equity', *Journal of marketing* **57**(1), 1–22.

Keller, S. & Meaney, M. (2017), 'Talent management for the twenty-first century'.
URL: *https://www.mckinsey.com/business-functions/organization/our-insights/attracting-and-retaining-the-right-talent*

KTH (2021), 'Sustainable development'.
URL: *https://www.kth.se/en/om/miljo-hallbar-utveckling/utbildning-miljo-hallbar-utveckling/verktygslada/sustainable-development/hallbar-utveckling-1.350579*

Lamkanfi, A., Demeyer, S., Giger, E. & Goethals, B. (2010), Predicting the severity of a reported bug, *in* '2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)', pp. 1–10.

Lehman, M. M. (1980), 'Programs, life cycles, and laws of software evolution', *Proceedings of the IEEE* **68**(9), 1060–1076.

Lewis, R. E. & Heckman, R. J. (2006), 'Talent management: A critical review', *Human Resource Management Review* **16**(2), 139–154. The New World of Work and Organizations.
URL: *https://www.sciencedirect.com/science/article/pii/S1053482206000271*

Li, Z., Lu, S., Myagmar, S. & Zhou, Y. (2006), 'Cp-miner: finding copy-paste and related bugs in large-scale software code', *IEEE Transactions on Software Engineering* **32**(3), 176–192.

Lievens, F. & Highhouse, S. (2003), 'The relation of instrumental and symbolic

attributes to a company's attractiveness as an employer', *Personnel psychology* **56**(1), 75–102.

Lievens, F. & Slaughter, J. E. (2016), 'Employer image and employer branding: What we know and what we need to know', *Annual Review of Organizational Psychology and Organizational Behavior* **3**(1), 407–440.
URL: *https://doi.org/10.1146/annurev-orgpsych-041015-062501*

Lunenburg, F. C. (2012), 'Organizational structure: Mintzberg's framework', *International journal of scholarly, academic, intellectual diversity* **14**(1), 1–8.

McCabe, T. J. (1976), 'A complexity measure', *IEEE Transactions on Software Engineering* **SE-2**(4), 308–320.

McKendrick, J. (2017), 'Technology is driving entrepreneurial growth, and we're not just talking about silicon valley'.
URL: *https://www.forbes.com/sites/joemckendrick/2017/11/28/technology-is-driving-entrepreneurial-growth-and-were-not-just-talking-about-silicon-valley/63b121917cd0*

McKinsey & Company (2016), 'Digital globalization: The new era of global flows'.

Michaels, E., Axelrod, E., Handfield-Jones, H. & Axelrod, B. (2001), *The War for Talent*, The War for Talent, Harvard Business School Press.

Midler, C. & Silberzahn, P. (2008), 'Managing robust development process for high-tech startups through multi-project learning: The case of two european start-ups', *International Journal of Project Management* **26**(5), 479–486.

Mintzberg, H. (1978), 'The structuring of organizations'.

Mintzberg, H. (1981), *Organization Design: Fashion or Fit*, Graduate School of Business Administration, Harvard University.

Moser, K., Tumasjan, A. & Welpe, I. M. (2015), Small, but attractive: the effect of employer branding and legitimacy on startup attractiveness, *in* 'Academy of Management Proceedings', Vol. 2015, Academy of Management Briarcliff Manor, NY 10510, p. 10528.

Oladapo, V. (2014), 'The impact of talent management on retention', *Journal of Business Studies Quarterly* **5**(3).

Peters, T. J. & Waterman, R. H. (1984), *In Search of Excellence: Lessons from America's Best-Run Companies*, Harper Row.

Picken, J. C. (2017), 'From startup to scalable enterprise: Laying the foundation', *Business Horizons* **60**(5), 587–595.
URL: *https://www.sciencedirect.com/science/article/pii/S0007681317300605*

Ray, B., Posnett, D., Filkov, V. & Devanbu, P. (2014), A large scale study of programming languages and code quality in github, *in* 'Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering', pp. 155–165.

Rothwell, W. (2010), *Effective succession planning: Ensuring leadership continuity and building talent from within*, Amacom.

Saunders, M., Lewis, P. & Thornhill, A. (2015), *Research Methods for Business*

Students, Pearson Education Limited.
URL: *https://books.google.se/books?id=vUdOCgAAQBAJ*

Schiemann, W. A. (2014), 'From talent management to talent optimization', *Journal of World Business* **49**(2), 281–288. Talent Management.
URL: *https://www.sciencedirect.com/science/article/pii/S1090951613000886*

Schöpfel, J. & Farace, D. J. (2010), Grey literature, *in* 'Encyclopedia of library and information sciences', pp. 2029–2039.

Schuler, R. S., Jackson, S. E. & Tarique, I. (2011), 'Global talent management and global talent challenges: Strategic opportunities for ihrm', *Journal of World Business* **46**(4), 506–516. Middle East Special Issue Section.
URL: *https://www.sciencedirect.com/science/article/pii/S1090951610000684*

Shah, S. & Corley, K. (2006), 'Building better theory by bridging the quantitative-qualitative divide', *Journal of Management Studies* **43**, 1821–1835.

Smart, B. (1999), *Topgrading : how leading companies win by hiring, coaching, and keeping the best people*, Prentice Hall Press, Paramus, NJ.

Spolsky, J. (2007), 'Finding great developers', *Smart and Gets Things Done: Joel Spolsky's Concise Guide to Finding the Best Technical Talent* pp. 19–39.

Stack Overflow (2020), 'Sustainable development'.
URL: *https://insights.stackoverflow.com/survey/2020*

Stahl, G., Bjorkman, I., Farndale, E., Morris, S., Paauwe, J., Stiles, P., Trevor, J. & Wright, P. (2016), 'Six principles of effectice global talent management', *IEEE Engineering Management Review* **44**, 112–119.

Sutton, S. M. (2000), 'The role of process in software start-up', *IEEE Software* **17**(4), 33–39.

Thunnissen, M. & Gallardo-Gallardo, E. (2017), *Talent Management in Practice: An Integrated and Dynamic Approach*.

Tom, E., Aurum, A. & Vidgen, R. (2013), 'An exploration of technical debt', *Journal of Systems and Software* **86**(6), 1498–1516.

Turban, D. B. & Cable, D. M. (2003), 'Firm reputation and applicant pool characteristics', *Journal of Organizational Behavior: The International Journal of Industrial, Occupational and Organizational Psychology and Behavior* **24**(6), 733–751.

Yin, R. K. (2017), *Case study research and applications: Design and methods*, Sage publications.

Yoffie, D. B. & Cusumano, M. A. (1999), 'Building a company on internet time: Lessons from netscape', *California Management Review* **41**(3), 8–28.

Zaharee, M., Lipkie, T., Mehlman, S. K. & Neylon, S. K. (2018), 'Recruitment and retention of early-career technical talent', *Research-Technology Management* **61**(5), 51–61.
URL: *https://doi.org/10.1080/08956308.2018.1495966*

Zhang, H., Zhang, X. & Gu, M. (2007), Predicting defective software components from code complexity measures, *in* '13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)', pp. 93–96.

TRITA ITM-EX 2021:240