

JYU DISSERTATIONS 514

---

**Kai-Kristian Kemell**

# Improving Software Development in Early-Stage Startups

---



UNIVERSITY OF JYVÄSKYLÄ  
FACULTY OF INFORMATION  
TECHNOLOGY

JYU DISSERTATIONS 514

---

**Kai-Kristian Kemell**

# **Improving Software Development in Early-Stage Startups**

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella  
julkisesti tarkastettavaksi yliopiston Ylistönrinteen salissa YAA303  
toukokuun 6. päivänä 2022 kello 14.

Academic dissertation to be publicly discussed, by permission of  
the Faculty of Information Technology of the University of Jyväskylä,  
in Ylistönrinne, auditorium YAA303, on May 6, 2022 at 14 o'clock.



JYVÄSKYLÄN YLIOPISTO  
UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2022

Editors

Marja-Leena Rantalainen

Faculty of Information Technology, University of Jyväskylä

Ville Korkiakangas

Open Science Centre, University of Jyväskylä

Copyright © 2022, by University of Jyväskylä

ISBN 978-951-39-9133-3 (PDF)

URN:ISBN:978-951-39-9133-3

ISSN 2489-9003

Permanent link to this publication: <http://urn.fi/URN:ISBN:978-951-39-9133-3>

## ABSTRACT

Kemell, Kai-Kristian

Improving Software Development in Early-Stage Startups

University of Jyväskylä, 2022, 106 p. + included articles

(JYU Dissertations

ISSN 2489-9003; 514)

ISBN 978-951-39-9133-3 (PDF)

Startup companies are important drivers of economic growth globally. Over the last two decades, software startups have become a part of mainstream culture, and have, in the process, become associated with innovativeness and various success stories. Many of the current and up-and-coming tech giants, the so-called unicorns with a valuation of over one billion USD, are examples of these startup success stories, some more well-known than others. However, past this illustrious image, the vast majority of startups fail, and in up to 98 % of new business ideas in general fail.

Software startups operate in a unique context often characterized by disadvantage that stems from various factors that vary by startup. This unique nature of the software startup context presents issues when it comes to applying existing knowledge of Software Engineering (SE) (or Information Systems Development (ISD)) into the startup context. Various research findings, existing SE/ISD methods, and lessons learned from practice come from more established software organizations such as multinational corporations. For example, Agile methods are more equipped to tell an organization 'how' to develop software in a situation where the needs of the customer are well understood. On the other hand, startups often operate in a situation where it is also unclear 'what' should be developed and there is no clear customer in sight yet.

This dissertation focuses on better understanding the software startup context in SE, with a focus on how software startups develop software. To this end, the dissertation ultimately proposes a method for early-stage software startups. The dissertation comprises five academic articles, out of which three are conference publications and two are journal publications. The articles utilize qualitative methods to approach the different issues in each article. The results of the dissertation further our understanding of how software startups work, and the method presented in the fifth and final article of the dissertation will ideally help early-stage startups work more systematically.

Keywords: startup, software startup, software engineering, software development method, software development practice, decision-making, the essence theory of software engineering

## TIIVISTELMÄ (ABSTRACT IN FINNISH)

Kemell, Kai-Kristian

Ohjelmistokehityksen parantaminen alkuvaiheen startup-yrityksissä

Jyväskylä: Jyväskylän Yliopisto, 2022, 106 p. + alkuperäiset artikkelit

(JYU Dissertations

ISSN 2489-9003; 514)

ISBN 978-951-39-9133-3 (PDF)

Startup-yritykset ovat maailmanlaajuisesti merkittäviä markkinavoimia. Etenkin edeltävän kahdenkymmenen vuoden aikana ohjelmistoalan startup-yrityksistä on tullut maailmanlaajuinen kulttuuri-ilmiö liiketoiminnan kontekstissa. Samalla startup-yritykset on alettu yhdistää innovaatioihin ja niihin lukuisiin onnistumistarinoihin, joita teknologia-alalla on viime aikoina nähty. Käytännössä kuitenkin valtaosa startupeista epäonnistuu ja uusista liiketoimintaideoista ylipäänsä jopa 98 % epäonnistuu.

Ohjelmistostartupit toimivat ainutlaatuisessa kontekstissa, jonka määrittävä tekijä ovat haasteet ja ongelmat. Startup-yritykset kohtaavat erilaisia haasteita ja ongelmia, kuten resurssien puute tai epävarmuus, jotka vaihtelevat startup-yritysten välillä. Tämän seurauksena startup-yritysten voi olla vaikea hyödyntää olemassa olevaa tietoa ohjelmistokehityksestä omaan tilanteeseensa. Olemassa oleva tutkimustieto, käytännön kokemuksista saadut opit ja nykyiset ohjelmistonkehitysmenetelmät, jotka ovat syntyneet suurten ohjelmistoyritysten kokemuksista ja vastaavat niiden ongelmiin, eivät välttämättä sovi startup-yritysten kontekstiin. Esimerkiksi ketterät kehitysmenetelmät (Agile) keskittyvät siihen, miten ohjelmistoja tulisi kehittää, kun asiakas on selvillä ja tiedetään jo mitä halutaan kehittää. Startup-yrityksen tilanne taas on usein se, että selvää asiakasta ei ole vielä tiedossa, eikä siitäkään ole selvää käsitystä, että millainen kehitettävän ohjelmiston tai palvelun pitäisi lopulta olla.

Tämä väitöskirja tutkii startup-yrityksiä ohjelmistokehityksen näkökulmasta. Väitöskirja keskittyy tutkimaan sitä, miten ohjelmistoalan startup-yritykset kehittävät ohjelmistoja. Tämä väitöskirja koostuu viidestä artikkelista, joista kolme on julkaistu tieteellisissä konferensseissa ja kaksi tieteellisissä lehdissä. Tutkimuksen tulokset auttavat meitä ymmärtämään paremmin, miten startup-yritykset kehittävät ohjelmistoja. Lisäksi väitöskirjan viidennessä artikkelissa esitellään menetelmä, jonka tarkoitus on auttaa aikaisessa vaiheessa olevia ohjelmistostartuppeja työskentelemään systemaattisemmin.

Avainsanat: startup, ohjelmisto-startup, ohjelmistotuotanto, ohjelmistokehityskäytänteet, päätöksenteko, ohjelmistotuotannon Essence-teoria, ohjelmistonkehitysmenetelmä

**Author**

Kai-Kristian Kemell  
Faculty of Information Technology  
University of Jyväskylä  
Finland  
ORCID 0000-0002-0225-4560

**Supervisors**

Pekka Abrahamsson  
Faculty of Information Technology  
University of Jyväskylä  
Finland

Tuure Tuunanen  
Faculty of Information Technology  
University of Jyväskylä  
Finland

**Reviewers**

Kari Smolander  
Department of Software Engineering  
Lappeenranta-Lahti University of Technology  
Finland

Jürgen Münch  
Department of Business Informatics  
Reutlingen University  
Germany

**Opponent**

Kieran Conboy  
School of Business & Economics  
National University of Ireland Galway  
Ireland

## ACKNOWLEDGEMENTS

To give these acknowledgments some further context, I want to note that I originally came to Jyväskylä a long time ago to study history, all the way back in 2009 (and it is, or was, now 2022). At the time, I moved to the Kortepohja student village for what I assumed would be 5 years, in order to get a master's degree. Now, almost 13 years later, I have finally managed to get out of Kortepohja, and I am leaving Jyväskylä with two master's degrees *and* a doctoral degree. It has been a long journey, and I certainly stayed in Kortepohja far longer than I thought I would. As such, these acknowledgments occasionally span further into the past than just the most recent four years that I have been working on this dissertation, as I am also saying goodbye to Jyväskylä in the process.

First and foremost, I want to extend the biggest thanks to main supervisor, professor Pekka Abrahamsson. I have deeply enjoyed working with Pekka every step of the way. Pekka has been a very supportive supervisor whose advice and positive feedback have been integral during these past four years. Pekka's style of supervising with me was rather hands-on, and as a result, I never felt particularly lost at any point. I always had some clear goals to work towards, either short-term or long-term ones.

I would also like to thank my second supervisor, professor Tuure Tuunanen. Tuure was the one who initially reached out to me while I was working as an intern at the department during my preceding studies, and asked me what I had planned on doing after graduation. Based on our conversation, Tuure introduced me to Pekka, which ultimately was the starting point for what you are now reading. Past this, Tuure helped me shape my argumentation, and helped me stay on track when it came to my goals. While the road was not always the straightest one, I like to think that I eventually reached the goal.

Further in relation to this dissertation process, I would like to thank both of my preliminary examiners, professor Jürgen Münch and professor Kari Smolander, for taking their time to review this dissertation and for their suggestions on how to improve it. In this final version, I have tried my best to address their comments in order to improve this dissertation.

Moving on to my colleagues, I must first thank Ville Vakkuri. We originally ended up working together as a result of a strict deadline necessitating my help on a paper. What was supposed to be just a little bit of help on one conference paper eventually became a close, long-term collaboration on AI ethics research. Over these past three or so years, I have very much enjoyed working with Ville. Thank you for all those marathon-length paper writing sessions, and for the ones to come.

I would also like to thank everyone at the JYU Startup Lab. I especially thank the Old Guard: Joonas Himmanen, Juhani Risku, and Johannes Impiö. I also thank the not-quite-but-almost Old Guard: Taija Kolehmainen and Joni Kultanen. I also thank all the newer team members, who I regrettably did not get to know so well due to the COVID-19-induced work-from-home situation that never quite seemed to end. The Startup Lab was a great community to be a

part of during the dissertation process, and I enjoyed my time at the office while it lasted. In the end, I spent more than half of my doctoral studies working remotely as a result of the early 2020 pandemic that is still on-going at the time of writing.

As for (other) fellow researchers, I thank all of my co-authors who have worked on papers with me so far, as well as the Software Startup Research Network that many of you are a part of. In particular, I thank Anh Nguyen-Duc for past, current, and future collaboration.

I also want to thank Tapio Tammi, the amanuensis of the (ex) department of Information Systems at JYU. Tapio was my first boss, and I felt like my time as an intern at the department was important to me in mentally graduating from being a university student and getting my foot in the door for future endeavors. In this vein, I would also like to thank Eetu Luoma, who regrettably is no longer with us but who was my second boss at the time.

Outside all the hours spent at the office or at home working on various papers and this dissertation, I thank my old friends from the history department. Thank you Zachris Haaparinne, Janne Tuomenlehto, Tuomas Olkkola, and Nooa Nykänen, for helping me keep my mind off my studies at times (and sometimes on them) during these 12 years. Kortepohja signing off. Viimeinen sammutti valot.

In this vein, I also thank my friends from further down south in Finland, or “the boys”: Heikki Hellman, Mikko Paavola, Toni Turunen, Heikki Mäkinen, and Tomi Lakianperä. I also extend this thanks to Eetu Pitkänen and Teemu Heikinheimo and extend a special thanks to Nick Hägerström. There are also many of you not mentioned by name, but all the same, thank you, too, for keeping me company all this time.

Similarly, I thank my wife Isis Kemell for keeping me company during this process. With the work-from-home situation dragging on for two years now, you have seen most of this happen up close. There is not much to say in that light but thank you for being there! I love you.

Finally, I thank my parents Sirpa Kemell and Ilpo Kemell for their support during my studies, and the past 31 years in general. While you were quite shocked about my initial decision to start studying history of all things at JYU back then, I think things turned out alright in the end.

In the end, I am already working at another university as a researcher. While I may, at last, be done with studying, I nonetheless seem to be staying in the university for another few years at the very least, and as such this dissertation is ultimately turning out to be just another step along the way. As it seems, I am certainly not done with writing papers for now.

Jyväskylä 06.05.2022  
Kai-Kristian Kemell



## LIST OF INCLUDED ARTICLES

- I Kemell, K.-K., Nguyen-Duc, A., Wang, X., Risku, J., & Abrahamsson, P. (2018). The Essence theory of software engineering : large-scale classroom experiences from 450+ software engineering BSc students. In PROFES 2018 : Product-Focused Software Process Improvement : 19th International Conference, Proceedings (pp. 123-138). Springer. Lecture Notes in Computer Science, 11271.
- II Kemell, K.-K., Ravaska, V., Nguyen-Duc, A., & Abrahamsson, P. (2020). Software startup practices : software development in startups through the lens of the Essence theory of software engineering. In PROFES 2020 : 21st International Conference on Product-Focused Software Process Improvement, Proceedings (pp. 402-418). Springer. Lecture Notes in Computer Science, 12562.
- III Vakkuri, V., Kemell, K. -K., & Abrahamsson, P. (2020). ECCOLA - a method for implementing ethically aligned AI systems. In Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, pp. 195-204.
- IV Nguyen-Duc, A., Kemell, KK. & Abrahamsson, P. (2021). The entrepreneurial logic of startup software development: a study of 40 software startups. *Empirical Software Engineering*, 26.
- V Kemell, K.-K., Suoranta, M., Nguyen-Duc, A., & Abrahamsson, P. (2022). A card-based method for early-stage software startups. Unpublished manuscript, to be submitted to a journal.

The author is the lead author in three of the papers and the second author in two of the papers. More specifically, in Article I, the author analyzed the data and produced the research publication. In Article II, the author designed the research and provided guidance in carrying out the data collection to the second author and wrote the article publication included in this dissertation. In Article III, the author jointly designed the research, collected the data, and wrote the publication with the first author. In Article IV, the author analyzed the data together with the first author and co-wrote the paper. In Article V, the author designed the research, gathered and analyzed the data, and wrote the publication.

## FIGURES

FIGURE 1.	Startup life-cycle according to Salamzadeh et al. (2015) .....	24
FIGURE 2.	Customer development process in software startups (Blank, 2013) .....	24
FIGURE 3.	Product development process in software startups (Blank, 2013) .....	24
FIGURE 4.	Startup learning process and product development process as adapted by Wang et al. (2016) .....	25
FIGURE 5.	The Greenfield Startup Model (Giardino et al., 2016).....	28
FIGURE 6.	Essence kernel alphas (Jacobson et al., 2012; Object Management Group, 2018) .....	41
FIGURE 7.	Card example from the method: startup card 2 .....	67
FIGURE 8.	Startup Cards 1 and 2 .....	98
FIGURE 9.	Startup Cards 3 and 4 .....	99
FIGURE 10.	Startup Card 5 and 6.....	100
FIGURE 11.	Startup Cards 7 and 8 .....	101
FIGURE 12.	Startup Cards 9 and 10 .....	102
FIGURE 13.	Startup Cards 11 and 12 .....	103
FIGURE 14.	Startup Cards 13 and 14 .....	104
FIGURE 15.	Startup Cards 15 and 16 .....	105
FIGURE 16.	Startup Card 17.....	106

## TABLES

TABLE 1.	Research approach overview of the included articles .....	45
TABLE 2.	Overview of the startup cards.....	66
TABLE 3.	Primary Empirical Contributions (PECs) of the dissertation .....	74

# CONTENTS

ABSTRACT

TIIVISTELMÄ (ABSTRACT IN FINNISH)

ACKNOWLEDGEMENTS

LIST OF INCLUDED ARTICLES

FIGURES AND TABLES

CONTENTS

1	INTRODUCTION .....	13
1.1	Motivation.....	13
1.2	Research Goals .....	15
1.3	Structure of the Dissertation .....	17
2	THEORETICAL BACKGROUND .....	19
2.1	Key Concepts.....	20
2.1.1	Conceptualizing Software Startups.....	20
2.1.2	The Startup Life-Cycle.....	23
2.1.3	Practice, Technique, and Method as Concepts .....	26
2.2	Software Development in Startups .....	27
2.2.1	Characteristics of Software Development in Startups .....	27
2.2.2	Agile Development in Startups.....	29
2.3	Startup Practices and Methods .....	31
2.3.1	Lean Startup.....	31
2.3.2	High-Profile Startup Practices.....	32
2.3.3	Growth Hacking.....	33
2.3.4	Research-Based Methods, Practices, and Tools for Startups ..	35
2.3.5	Suitability and Relevance of Existing SE Practices.....	36
2.4	The Essence Theory of Software Engineering .....	37
2.4.1	The Essence Language.....	38
2.4.2	The Essence Kernel .....	40
2.4.3	Essence in Research .....	41
3	RESEARCH METHODOLOGY .....	43
3.1	Research Evolution.....	43
3.2	Research Approach.....	44
3.3	Case Study .....	46
3.4	Action Research.....	47
3.5	Data Collection and Analysis Methods .....	48
3.5.1	Qualitative Interview.....	48
3.5.2	Thematic Analysis.....	50
4	OVERVIEW OF THE ARTICLES.....	53

4.1	Article I: The Essence Theory of Software Engineering – Large-Scale Classroom Experiences from 450+ Software Engineering BSc Students .....	53
4.2	Article II: Software Startup Practices – Software Development in Startups Through the Lens of the Essence Theory of Software Engineering.....	55
4.3	Article III: ECCOLA – A Method for Implementing Ethically Aligned AI Systems.....	56
4.4	Article IV: The Entrepreneurial Logic of Startup Software Development – A Study of 40 Software Startups.....	58
4.5	Article V: Startup Cards – A Method for Early-Stage Software Startups .....	59
5	RESULTS AND CONTRIBUTIONS .....	61
5.1	Results .....	61
5.1.1	The Essence Theory of Software Engineering in a Student and Startup Context.....	61
5.1.2	Work Practices and Decision-Making in Software Startups...	63
5.1.3	Method: Startup Cards for Early-Stage Startups.....	65
5.2	Validity Threats.....	68
5.2.1	Article I .....	68
5.2.2	Article II.....	69
5.2.3	Article III.....	70
5.2.4	Article IV .....	71
5.2.5	Article V.....	72
5.3	Contributions.....	73
5.3.1	Theoretical Contributions .....	74
5.3.2	Practical Contributions.....	78
5.3.3	Limitations .....	81
5.3.4	Future Research Suggestions.....	82
	YHTEENVETO (SUMMARY IN FINNISH) .....	83
	REFERENCES.....	86
	APPENDIX: STARTUP CARDS FOR EARLY-STAGE STARTUPS.....	97
	ORIGINAL PAPERS	

# 1 INTRODUCTION

This section outlines the key concepts of this dissertation and discusses the primary motivation behind the research included in it (Sections 1.1 and 1.2), which are then further elaborated on in the background section (Section 2). Additionally, this section contains the content outline of the dissertation (Section 1.3).

## 1.1 Motivation

This dissertation is about software development in the specific context of software startups. Startups, in the past decade especially, have become associated with success stories (Giardino, Wang & Abrahamsson, 2014). like those of some of the newer multinational technology corporations such as Spotify and Facebook. Startups are often seen as disruptors that challenge existing market leaders with novel ideas and explosive growth, overtaking the old and inflexible companies operating in the area, or forcing them to innovate to fight back.

Startups are important drivers of economic growth globally. There are currently more than 140000 startups in Europe, and roughly a third of these have managed to acquire at least one round of funding (The State of European Tech 2020). In total, this adds up to some €43,3 billion invested in European tech startups in 2019 (The State of European Tech 2020), with similar projected numbers for 2020 despite the unforeseen effects of the pandemic year. US startups, on the other hand, saw investment up to 140 billion \$USD in 2019 (PitchBook, 2019).

Yet behind the success stories and the impressive numbers, the vast majority of startups end in failure (Crowne, 2002; Blank, 2013; Giardino et al., 2014c), much like how most new companies in general fail. In terms of new product ideas in general, over 98% fail (Mullins & Komisar, 2009). Thus, much of the invested capital is wasted. This has set the stage for academic research

looking to better understand startups in order to, ideally, prevent at least some of these failures.

Today, startups are studied across disciplines, including Software Engineering (SE), and to some extent IS (Information Systems). The construct 'startup' has been defined differently across these various disciplines startups are studied in (Sutton, 2000; Ries, 2011; Blank, 2013; Unterkalmsteiner et al., 2016; Ghezzi, 2018; Steininger, 2019). For the purposes of this introduction section, we can highlight the core aspect of startups: startups are temporary organizations. A startup either eventually becomes a mature organization, or it fails somewhere along the way, as most startups do. During this stage of their life cycle, these companies differ from other types of software organizations. This serves as the motivation for many startup-related studies. As Unterkalmsteiner et al. (2016) remark: "software startups are quite distinct from traditional mature software companies, but also from micro-, small-, and medium-sized enterprises, introducing new challenges relevant for software engineering research."

Due to this unique context startups operate in, the key issue, from an academic point of view, is the applicability of existing research findings. If startups differ from other types of companies, to what extent can we apply the findings of past studies focusing on these traditional companies into the software startup context? While the exact nature of this uniqueness remains a topic of discussion in startup research in SE, it has become a common motivation behind startup research in the area. Moreover, highlighting the importance of SE factors in software startups in particular, Klotins et al. (2019) point out that "inadequacies in software engineering could be a significant contributing factor to the high start-up failure rate and precede any marketing or business-related challenges."

In this regard, startups are known to seldom utilize existing SE methods. Especially earlier on in their lifecycles, software startups largely develop software using various singular Agile practices (Paternoster et al., 2014) as opposed to using textbook methods. Consequently, popular research areas and industry trends in SE may not always have much relevance to startups. One cannot do Agile at scale when there is no scale, or DevOps when there is only one small team to work with and no silos to break down. Yet startups, like any other type of software organization, should concern themselves with structuring their work processes (Ries, 2011).

This difference between startups and more mature software organization is the second core motivation behind this thesis. Software startups operate in a unique context characterized by its uncertainty and even chaotic nature (Ries 2011, Paternoster et al., 2014), facing challenges that can differ greatly from those larger companies face (Giardino et al., 2015). Startups find it difficult to utilize existing software engineering methods that have been devised with larger, more mature organizations in mind (Paternoster 2014). As Bosch et al. (2013) aptly remark "[Agile methods] are mainly applied in situations where the problem is fairly well understood but the solution is not. In a startup context, however, neither the problem nor the solution is well understood." Further studies that

help us understand these differences between software startups and mature software organizations are needed, as are studies that look at methods from the point of view of startups.

The primary motivation behind this dissertation is the high failure rate of startups. Though software startups are often associated with success stories (Giardino, Wang & Abrahamsson, 2014), the vast majority of software startups ultimately end in failure (Crowne, 2002; Blank, 2013; Giardino et al., 2014). Though most new companies in general fail, existing studies have argued that startup failures commonly attributed to business model issues may in fact be closely related to software development issues (Klotins et al., 2019). The importance of product development issues in relation to startup failures is also suggested by Crowne (2002) in an early startup research paper. In the paper, Crowne (2002) discusses various potential product development issues across startup stages, although with no supporting empirical studies. Unterkalmsteiner et al. (2016) also remark that “inadequacies in applying engineering practices could be a significant contributing factor to startup failure.” Working on the wrong product and failing to sufficiently validate the idea is a common cause for failure as well, where requirements engineering (or validation) can be seen as a key failure cause related to SE (Bosch et al., 2013).

Existing research has emphasized the importance of better understanding SE in software startups and providing better support for SE in startups (Unterkalmsteiner et al., 2016; Pantiuchina et al., 2017; Bajwa et al., 2017). If startups struggle to utilize the results of existing studies, and struggle to make use of existing methods and practices aimed at larger, more established software companies, studies looking at methods and practices specifically in the startup context may alleviate some of the problems faced by startups.

## 1.2 Research Goals

The primary objective of this dissertation is to create a method to help startups tackle their key challenges. These are highlighted in existing publications, namely that of Wang et al. (2016), as well as in that of Klotins et al. (2019). In brief, startups struggle primarily with: (1) building product, (2) customer acquisition, (3) funding, (4) building the team, and (5) business model. Yet, while these may intuitively seem more related to business than SE, business and SE are closely intertwined in startups. Klotins et al. (2019) expand mainly on the first of these three challenges, and, as Klotins et al. (2019) also argue, many seemingly business-related issues can in fact stem from SE problems in software startups.

Because of its article-based format, this dissertation has multiple objectives building up to the goal of creating this method for software startups. On a general level, the aim of this dissertation has been to further our understanding of how software startups work (Articles II and IV) as well. As a research question, this is summarized as follows:

RQ How can we improve software development in startups?

In the process of answering this question, the articles included in this dissertation tackle the following, more specific research objectives:

**Objective 1.** To evaluate the suitability of the Essence Theory of Software Engineering for small and immature software organizations, such as software startups (Article I).

**Objective 2.** To further our understanding of how software startups develop software, with a focus on practices (Articles II and IV) and decision-making (Article IV) in particular.

**Objective 3.** To develop a method for software startups, with a focus on tackling key challenges and antipatterns in startups (Articles III and V).

**Objective 4.** To evaluate the kernel of the Essence Theory of Software Engineering in the startup context (Article II).

Initially, when I first started working on this dissertation, the main objective was to evaluate the suitability of the Essence Theory of Software Engineering for the software startup context, and to then create a version of Essence better suited for it, if needed. Articles I and II supported this goal. In addition to studying what practices are common in software startups and devising a list of such practices, Article II evaluated how these practices fit Essence, and whether additional alphas (see Section 2.5 for further information on Essence) were required to accommodate them.

However, as my research progressed, I began to think, based on the results we were seeing, that Essence was not well-suited for software startups. It added unnecessary complexity to method adoption. Essence itself was difficult to adopt, and to use a method described with the Essence language, one had to learn to use Essence first.

The initial problems surfaced in the study in Article I where a large number of SE student teams utilized Essence for a practical course project. Though the teams also had positive sentiments about Essence, overall, the teams considered difficult to understand. At the time, we nonetheless kept pursuing this approach. In Article III, we develop a method for AI Ethics. Originally, this method was to be a card-based method described using the Essence language, much like the software startup method in Article V. These method development endeavours of Articles III and V contributed to each other through shared lessons learned due to the similar approach. Both methods were developed iteratively, using an Action Research (AR) approach, and were originally intended to be described using the Essence language.

The development of the methods of Articles III and V proceeded in tandem between 2018 and 2021. Early on, Essence already began to seem like an obstacle to the adoption of the AI Ethics method in Article III. The key issue was that, to utilize the method described using Essence, its users would first have to learn Essence. In other words, *in addition to* adopting a new method, its users would have to *also* learn Essence. Given that Essence was, based on Article I, also



considered difficult to learn on its own, we saw this as a problem. Failing to understand Essence despite trying to do so made it difficult for the teams to utilize the method correctly. These early versions of the AI Ethics method were tested with student teams who had been tasked with studying Essence through a course assignment and had nonetheless struggled to do so. These issues are also discussed in more detail in Article III (and in its extended journal version, Vakkuri, Kemell et al. (2021)).

Based on these lessons learned from the AR process of Article III and the results of Article I, Essence was ultimately not used to describe the Startup Cards in Article V. This has also resulted in the role of Essence, overall, being smaller than originally intended past Article II. Despite this being the case, though, the core philosophy behind Essence, i.e., essentializing SE practices to create method, was still utilized to devise the method in both Articles III and V, as we discuss in more depth in the papers themselves. The Startup Cards also utilize some notational characteristics of the Essence language, but do not fully formally utilize it anymore, so as to make the method easier to adopt for teams that are unfamiliar with the language – which, given the lack of widespread practitioner adoption of Essence (SEMAT, 2018), most teams arguably are, especially in startups. The cards could still be formalized using the Essence language, and the dissertation nonetheless presents different contributions related to Essence as well.

In summary, Articles I-IV further our understanding of how software startups develop software (Articles II & IV) or otherwise contributed to the creation process of method presented in Article V (Articles I & III).

### **1.3 Structure of the Dissertation**

This dissertation comprises five scientific publications: three conference articles and two journal articles. Together, these articles contributed to the creation of the fifth and final article, and with it the method proposed in it. Aside from the creation of the method, the articles, as discussed in the preceding subsection, contributed to our understanding of how software startups work.

The rest of this dissertation, leading up these articles, is structured as follows. Section 2 presents the theoretical background of this dissertation. Section 3 discusses the research methodologies utilized in the articles included in it. Section 4 presents more detailed paper summaries for each included article. Section 5 summarizes the results, threats to validity, and contributions of this dissertation. After this, the five articles included in this dissertation are presented in order. These articles are summarized briefly below, and in more detail in Section 4.

In Article I, we deploy the Essence Theory of Software Engineering, a tool aimed at established software organizations, in a student project setting in a practical course on Software Engineering. The study aims to understand whether inexperienced developers with little working history (as a team) could adopt and

utilize the tool successfully. As the method in Article V was originally going to be described using the Essence language, we first wanted to understand whether Essence could also be suitable for early-stage startups and early stage startup-like environments such as student project teams.

Article II is focused on the use of practices in software startups. The goal of the study is to propose an extensive list of different practices utilized by software startups. This is done by validating an existing list of practices with empirical data while adding new practices to the list as they emerged from the data. Additionally, the paper looks at these practices through the lens of the Essence Theory of Software Engineering. In doing so, we wish to understand whether startup practices also fit the framework of Essence, or whether some modifications of the framework would make it better suited for the software startup context.

In Article III, we develop a card-based method for AI ethics, using an iterative AR approach. The method, like the one in Article V, was originally intended to be described using the Essence language. Given the similar research approach and method design philosophy between the two articles and methods, the lessons learned from the development of this method directly contributed to Article V and its method – and vice versa. As these method development endeavours ran in tandem between 2018 and 2021, the opposite is also true, with the lessons learned from the method of Article V also affecting that of Article III.

Article IV studies decision-making in software startups. Using a theoretical framework from business studies, we look into how software startups make decisions. We characterize decision-making in software startups and propose a typology of software startups based on their decision-making logics.

Article V presents the main contribution of this dissertation: the Startup Cards. The Startup Cards consist of various key practices for software startups, formulated based on existing literature. These practices are intended to act as a remedy against the key challenges commonly faced by software startups and the anti-patterns commonly seen in them.

## 2 THEORETICAL BACKGROUND

This section presents the theoretical background of this dissertation. In Section 2.1, I discuss the key concepts of this dissertation in more detail. Section 2.1 provides an overview of software startups, as well as concepts related to methods and practices. The concept of *startup* and the characteristics associated with startups remain a topic of discussion in software startup research. As the driving force behind startup research is that startups differ from other business organizations (Unterkalmsteiner et al., 2016), understanding the nature of these differences, on a conceptual level, is relevant. The section also looks at various models describing the life cycle of startups, which is relevant as this dissertation is about *early-stage* software startups.

Section 2.2 then moves the focus from (software) startups in general to software engineering in startups, which is the main research area of this dissertation. Startups also differ traditional business organizations in terms of how they develop software. Startups prefer different development practices and seem to struggle to utilize existing software engineering methods due to their unique context.

Some practices and methods designed for (or created in) this startup context exist. These are discussed in Section 2.3. As these are not abundant, the section also includes discussion on *growth hacking*, a digital marketing strategy favored by startups that is often connected with software engineering activities. The subsection also includes discussion on research-based initiatives relevant in this regard.

Finally, Section 2.4 introduces the Essence Theory of Software Engineering. Essence provides a way of modeling practices and methods in software engineering. We originally planned on utilizing Essence to describe the method presented in Article V, the creation of which was the main objective of the dissertation. However, as is discussed in detail later in this dissertation, due to our research findings and experiences with using Essence, the role of Essence became smaller as work on this dissertation progressed. Nonetheless, Essence is evaluated in Article I, acts as a framework and is further evaluated in Article II, and was used in the process of developing the methods in Articles III and V. It

therefore still retains a notable role in this dissertation, which is why it is discussed here.

## 2.1 Key Concepts

This dissertation is focused on software development in startups. While speaking of *startups*, I speak of a specific subset of startups: software startups. Both *startup* and *software startup* as concepts are clarified in this section. As is soon highlighted, startups are *temporary* organizations. No startup remains a startup indefinitely. Ultimately, the startup either fails or becomes a mature company. As such, while discussing startups, this section also looks at the startup life cycle. Finally, the third subsection focuses on key concepts related to describing work in the context of software engineering, including the concepts of *method* and *practice*.

### 2.1.1 Conceptualizing Software Startups

In terms of academic research, startups are studied across disciplines. Being companies, they have been studied in economic disciplines and as part of organizational research in various disciplines, including IT ones. The term “startup” has been defined differently across various disciplines (Sutton, 2000; Ries, 2011; Blank, 2013; Unterkalmsteiner et al., 2016; Ghezzi, 2018; Steininger, 2019). New Technology-Based Firm (NTBF) is a concept used to discuss startups in many business and organizational research papers (cf. Donckels & Segers, 1990; Fudickar & Hottenrott, 2019). Given the tech-oriented nature of startups, Software Engineering (SE) and Information Systems (IS) scholars have also taken an interest in startups. However, IS papers focused on startups are few and far between in the top IS journals such as Management Information Systems Quarterly (MISQ). On the contrary, a large number of startup papers has been published in SE venues, where software startup research has become a well-established research area (Unterkalmsteiner et al., 2016). As startups are even associated with technology, and more specifically Information Technology (IT), by definition in some cases, it is not surprising that there is by now an extensive number of papers on software startups in SE.

According to Unterkalmsteiner (2016), software startup research in SE and IS dates back to 1994, when Carmel (1994) first introduced the concept of software startup (in the form of *software package* startup in that paper). Since then, startups have been studied from a multitude of point of views in SE, including studies on software engineering practices, as well as more business-oriented studies (Unterkalmsteiner et al., 2016). Business and SE are often closely intertwined in the startup context, resulting in various SE studies on software startups discussing business aspects to varying extents as well. Arguing for the importance of SE from a business point of view in startups, Klotins et al. (2019) remark that “inadequacies in software engineering could be a significant

contributing factor to the high start-up failure rate and precede any marketing or business-related challenges.”

The key argument behind all software startup research is that software startups are somehow different from conventional software companies. As briefly touched upon in the introduction section, “software startups are quite distinct from traditional mature software companies, but also from micro-, small-, and medium-sized enterprises, introducing new challenges relevant for software engineering research.” (Unterkalmsteiner et al., 2016). The exact nature of these differences is an on-going topic of discussion in the area, with different papers focusing on different characteristics. Paternoster et al. (2014) list 15 such characteristics software startup research has associated with software startups to differentiate them from other types of software companies: (1) highly reactive, (2) innovation, (3) uncertainty, (4) rapidly evolving, (5) time-pressure, (6) third party dependency, (7) small team, (8) one product, (9) low-experienced team, (10) new company, (11) flat organization, (12) highly risky, (13) not self-sustained, (14) lack of resources, and (15) little working history.

Aside from differentiating startups from other software organizations, these characteristics are occasionally used to define startups as well. However, whether the presence of any of these characteristics is required for a company to be considered a startup remains debatable. As there is no widely agreed-upon definition for what a startup is in software engineering literature, different papers may place emphasis different characteristics.

Perhaps the most commonly utilized definition of a startup is that of practitioner expert Steve Blank. Blank (2013) considers a startup “a temporary organization designed to search for a repeatable and scalable business model.” Thus, when a startup finds and successfully implements a sustainable business model, it ceases to be a startup and grows into a mature company. A startup, therefore, is a temporary organization that either becomes an established company or fails somewhere along the way, although it can still be challenging to determine when exactly a startup can be considered to have ceased to be a startup.

Blank’s definition is especially popular among practitioners (e.g., Baldrige & Curry 2021), but is also cited in various software startup research papers, as underlined by Unterkalmsteiner et al. (2016) utilizing Blank’s definition in their research agenda paper for the research area in SE. This dissertation also utilizes this definition. However, this definition was not directly used for data collection purposes in the articles included in this dissertation. For data collection purposes, given the lack of a widely agreed-upon definition for the concept, Articles II and V included any company whose founders considered it a startup. The study of Article IV, on the other hand, utilized certain criteria for including or excluding case companies, as discussed in more detail in the article.

In addition to the general definition of a startup, it is in order to briefly discuss what a software startup is. Many associate startups with software (and hardware) by definition and consider them technology companies by nature. One example of this is the way NTBF is considered synonymous with startup. To

nonetheless be more specific, in this dissertation, software startups are not exclusively startups whose main business is developing software. Rather, a software startup is any startup that delivers, creates, or captures value through software. For example, while Uber is a transportation company, its entire business ultimately revolves around its use of software to deliver value.

Returning to the fifteen characteristics summarized by Paternoster et al. (2014), it could be summarized that startups are inherently associated with disadvantage that stems from market, financial, or technological adversities (Wang & Nandhakumar, 2017). More specifically, this disadvantage stems from some of these fifteen characteristics. The exact nature of this disadvantage remains a topic of discussion in startup research, and it can arguably vary between startups. Some startups, for example, are more susceptible to having their ideas copied by other businesses than others due to the nature of their business idea. Altogether this results in a context that is unique to startups, differentiating them from mature software organizations (Unterkalmsteiner et al., 2016).

Finally, many of the characteristics discussed in this section are in some ways related to the temporary nature of software startups. As established, startups are considered to be temporary organizations. In practice, this temporary nature largely stems from the financial aspects. According to Blank (2013), a startup is looking for “a repeatable and scalable business model,” and so, in other words, a startup does not have one yet. Until a startup finds one, it can only keep going as long as it has capital to burn. This can be investor capital or the personal capital of the founder(s).

This is a widely acknowledged situation that has its own terminology and metaphors. Startup practitioners often use aviation metaphors to discuss this. A startup is seen as a plane that is on the runway, trying to take off. The runway, in this metaphor, presents the capital the startup has. As it is constantly burning capital in search of a viable business model, the length of the runway is determined by how much time it has until it runs out of capital. The take-off, then, is the point where the startup starts being able to support itself when its business, or, so to say, takes off.

This has various implications for startups in practice. Primarily, this situation creates time pressure. For example, time pressure be related to reaching the market as soon as possible, or finding the next source of funding in order to extend the runway. Time pressure, in turn, results in a need or urge to cut corners, which in SE often manifests as technical debt (which I discuss in more detail in Section 2.2). A startup may also feel the need to provide itself with a temporary source of income outside its (planned) core business. For example, it is not uncommon for startups developing software to take on externally commissioned projects as a source of side income. While this can provide the startup with capital, it can also make it difficult to make timely progress on their own product. In this fashion, the temporary nature of startups is directly related to at least four of the characteristics discussed by Paternoster et al. (2014): time pressure, uncertainty, not self-sustained, and lack of resources.

### 2.1.2 The Startup Life-Cycle

As established thus far, startups are temporary organizations. They either fail somewhere along the way, which most startups do, or successfully become mature companies. As this dissertation is focused on early-stage startups in particular, these life cycle models act as a framework for defining what an early-stage startup is in practice. To this end, many models exist for conceptualizing this startup journey from an initial idea to a startup, and from a startup to a mature organization.

One example of a life-cycle model proposed by an extant study is the one Nguyen-Duc et al. (2016) propose. This model splits the startup lifecycle into three phases:

1. Pre-startup stage: ideas are developed and need to be validated, startups in the quest for financial and human resources. Startup activities are carried out by founders or short-term hires. The purpose of this stage is to demonstrate business feasibility, team building and management. The common financing model is bootstrapping, family, friends and foes (FFF).
2. Startup stage: prototypes are developed and experimented, startups have already figured out the problem/solution match. Some revenue is generated, but not necessarily over the break-even point. Founder seeks support mechanisms from startup ecosystems, learn to accelerate their business development. The common financing model is own funding and seed funding.
3. Post-startup stage: products are extended, startups achieve the product/market match. Startups expand their customer bases, the revenue models are predictable and scalable. A hierarchical structure is formed within the startups. The common funding model is Series A, Series B, and other series.

Passaro et al. (2016) propose another model. Their model splits the startup lifecycle into four phases: (1) ideation, (2) intention, (3) start-up, and (4) expansion. In the ideation phase, the idea is generated and evaluated, with idea viability being the milestone that ends the phase. In the intention phase, an initial commitment is made towards carrying out the idea and a team is put together and further validation is carried out. The second phase ends in a prototype. In the third phase, the start-up phase, a business is built around the idea. This phase is focused on searching for funding and further product and business development and culminates in the first invoice. Finally, in the expansion phase, the startup begins to scale and eventually stops being a startup upon success. This model shares many similarities with the model of Nguyen-Duc et al. (2016). Notably, though, in both models the startup phase or stage is in the middle of the lifecycle, with the earlier steps being categorized under some other denomination.

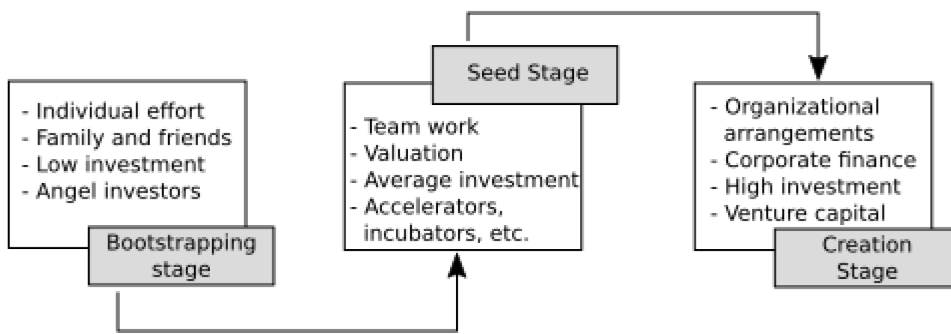


FIGURE 1. Startup life-cycle according to Salamzadeh et al. (2015)

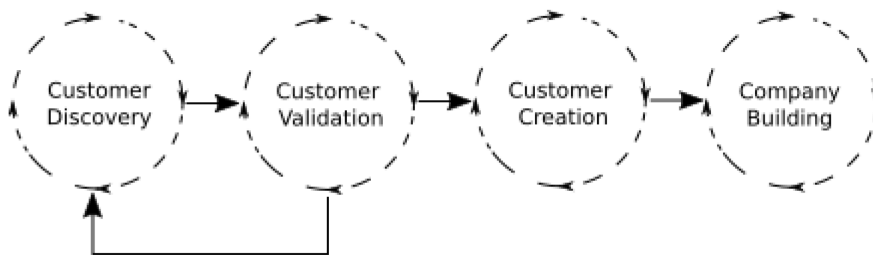


FIGURE 2. Customer development process in software startups (Blank, 2013)



FIGURE 3. Product development process in software startups (Blank, 2013)

A slightly different three-stage model is proposed by Salamzadeh et al. (2015) (Figure 1). This model has three stages: bootstrapping, seed, and creation stage. The first one is characterized by individual effort and low investment, largely from 3Fs (Family, Friends, Fools) or angel investors. The second stage is characterized by team work and participating in startup ecosystems such as accelerators and incubators, and in terms of finances, average investment. The third and final stage is about growing into a mature organization.

Blank (2013) considers learning to be central in any startup. Blank (2013) highlights two distinct processes that startups should go through simultaneously. First is the product development process that depicts, as the name implies, stages in product development (Figure 2). Secondly, in addition to the product development process, the startup engages in, or should engage in, a learning process where the business is developed: the customer development process (Figure 3). The customer development process is key in avoiding a situation where one develops a product no one wants to use. As Blank (2013) considers startups to be temporary organizations, where the primary objective of a startup is to find a business model, this learning process is related to establishing a viable and scalable business model in order for the startup to eventually grow into a mature organization.



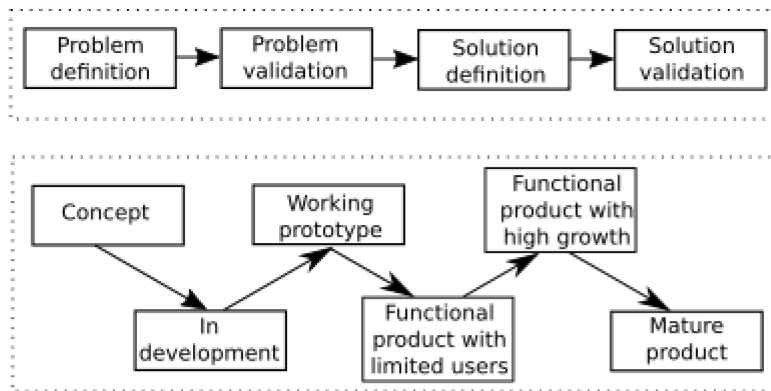


FIGURE 4. Startup learning process and product development process as adapted by Wang et al. (2016)

Wang et al. (2016) present a modified version of these two processes. This version is seen in Figure 4. These two processes are used to position this dissertation, especially in terms of Article V. The method Article V presents, the Startup Cards, is aimed at the highlighted areas of the process, whereas the faded-out parts are increasingly out of scope of the method. It is primarily intended to support the learning process and the earlier parts of the product development process, with the intent that the learning be prioritized early on.

Through the lens of the model of Nguyen-Duc et al. (2016), the model is most related to the *pre-startup stage*. The model associates many of the early-stage startup activities, such as idea validation, with this stage. On the other hand, the *startup stage* in this model is characterized by already having initial customers and some revenue. Most of the practices remain relevant in the *startup stage* as well, even though the focus of the method is on the preceding phase. Similarly, in the life cycle model of Passaro et al. (2016), the focus is on the ideation and intention phases, and to some extent the startup stage. Finally, from the point of view of the model of Salamzadeh et al. (2015), the focus would be on the bootstrapping stage, and to some extent the seed stage.

Aside from the method discussed in Article V, this dissertation has not specifically focused on early-stage startups. In both Articles II and IV, we utilized empirical data from startups. In selecting the case startups for these studies, the focus was *not* on early-stage startups. In fact, in Article IV, as I discuss in more depth later in this dissertation, we selected startups that were already further along with their product development, so as to be able to focus more on SE activities.

Having now discussed what software startups are, and having also discussed what *early-stage* startups are, the following sections shift the focus towards SE in startups. In the final subsection of this key concepts section, I discuss the key concepts related to methods in SE and IS. Afterwards, having discussed all the key concepts, I proceed to discuss software development in startups.

### 2.1.3 Practice, Technique, and Method as Concepts

The focus of this dissertation is on SE in software startups. Before discussing SE in the startup context, it is in order to briefly discuss the relevant concepts used to describe work in the context of SE.

Traditionally, failures in software development have been attributed to the use of irrational development approaches both by academics and practitioners. From this follows that SE/IS Development (ISD) methods are considered one solution to this problem (Fitzgerald 1996). How software is developed in practice is a topic studied widely in both Information Systems (IS) and Software Engineering (SE). In fact, SE by definition is about "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (SEVOCAB, n.d.).

A method describes a process. In IS, a method is a "a predefined and organized collection of techniques and a set of rules which state by whom, in what order, and in what way the techniques are used to achieve or maintain some objectives." (Tolvanen, 1998). Software development or SE methods, at their core, are structured processes designed for software development. They describe how work should be carried out in a software project, and utilize either natural or formal language, or some mix thereof, to do so.

A technique in IS, is a more atomic description of a work process. According to the above definition, a method consists of various techniques which describe smaller work processes, and together form a method. According to Tolvanen (1998), a technique is "a set of steps and a set of rules which define how a representation of an IS is derived and handled using some conceptual structure and related notation." In SE literature, practice is largely synonymous to method in the SE discourse. Jacobson et al. (2012) similarly remark that methods consist of practices. In addition, tools are software (and occasionally physical objects) used to carry out development in practice, such as Integrated Development Environments. Methods and practices may recommend using specific tools to carry out the prescribed work processes as well.

From here on out, this dissertation will use the established SE concepts to discuss these topics, e.g., practice over technique. While this is formally an IS dissertation, the papers included in it have been published in SE venues, and consequently utilize SE concepts. As such, I will speak of practices rather than techniques in this dissertation as well. To this end, what is SE in the field of SE is referred to as Information Systems Development (ISD) in the field of IS. These are not fully interchangeable, and as such I speak of SE in this dissertation rather than ISD, even if what is being studied here could also be discussed as ISD in IS publications.

Most methods have historically been in-house methods (Bubenko, 1986; Grant et al., 1992). This is also the case today, with in-house methods being common, but with out-of-the-box methods also still seeing some utilization (Ghanbari, 2017). This situation is highlighted in the recent article of (Kuhrmann et al., 2021), who argue that, despite decades of Agile, we still do not have a clear

idea of what really constitutes Agile development due to the myriad of different practices being mixed and matched to create various methods their creators consider Agile. A practical example of this phenomenon is the idea of ScrumBut. ScrumBut is a concept created to describe the various situations where organizations utilize Scrum, a SE method, but ultimately end up specifying some ways in which they deviate from the original method: "We use Scrum, but..."

While method tailoring is commonplace (Jacobson et al., 2012), method use is nonetheless seemingly norm in established software organization (Digital.ai 2020), even if they are not being used strictly by the book. Jacobson et al. (2012) even argue that using methods by the book is not something to strive for in the first place, and that methods should be tailored to best suit the project context at hand. To this day, though, in startups, and especially in newer and smaller software startups, less systematic software development continues to be common (Paternoster 2014), as is discussed later in this dissertation.

## 2.2 Software Development in Startups

As established thus far, the main argument behind startup research is that startups differ from other types of companies, which makes research findings concerning traditional companies not fully applicable to them. This is also the case in SE research. To understand *how* exactly (and if) software startups differ from other software organizations, various existing studies have looked at the state of practice of SE in software startups.

### 2.2.1 Characteristics of Software Development in Startups

Various existing studies have looked at different facets of SE in startups. As has already been briefly discussed in the introduction, startups typically utilize singular Agile practices and seldom use SE methods, especially in the earlier stages (Paternoster et al., 2014). The Greenfield Startup Model (Figure 5) (Giardino et al., 2016) presents one way of conceptualizing software development in startups. The model highlights key characteristics in software development in startups. The starting point is that startups operate under a notable lack of resources, which is often discussed in extant literature (e.g., Berg et al, 2018; Paternoster et al. 2014; Wang et al. 2016). The other characteristics in the model, such as the importance of the team (e.g., Cooper et al., 1994; Kemell, Elonen et al., 2020; Seppänen et al., 2017; Seppänen, 2020) and Technical Debt (e.g., Apa et al., 2020; Besker et al., 2018), are explored in the startup context various other studies.

Technical debt is "a metaphor for immature, incomplete, or inadequate artifacts in the software development lifecycle that cause higher costs and lower quality in the long run" (Seaman & Guo, 2011). As quality is seldom a focus in software startups (Klotins et al., 2019) and startups prioritize speed (e.g., as a result of time-to-market pressure, or due to the aforementioned lack of resources etc.), short-term gains are often prioritized over long-term ones. This is not

necessarily counter-productive: the technical debt will never have time to realize over the long-term if the startup fails, and most startups fail. Those startups that do last longer, however, have to address their technical debt later. In practice, this can mean simply abandoning old components or systems riddled with technical debt instead of attempting to refactor or restructure them (Article IV). As the Greenfield Model (Figure 1) (Giardino et al. 2016) posits, being forced to deal with the technical debt accumulated early on, or failing to deal with it, may hinder later performance.

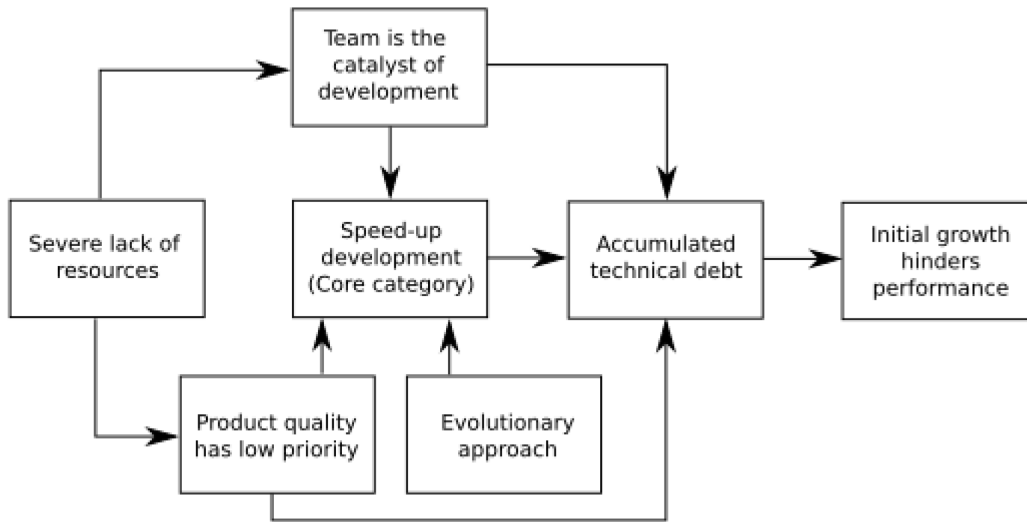


FIGURE 5. The Greenfield Startup Model (Giardino et al., 2016)

Startups largely focus on one product or service (Paternoster et al., 2014; Berg et al., 2018). With the entire business of the company based on that one development endeavor, business activities become closely intertwined with SE. What are perceived as business problems in software startups can in fact be SE problems under the surface (Klotins et al., 2019).

Another example of a model conceptualizing software development in startups is the Startup Hunter-Gatherer Cycle Model of Nguyen-Duc et al. (2015). Based on the Cynefin framework (Snowden & Boone, 2007) and the work of Steinert & Leifer (2012), the model splits activities in software startups into two categories: hunting and gathering. In this model, hunting activities are related to idea generation, requirements elicitation, and market and customer development. Gathering activities, on the other hand, are related to describing requirements, implementing prototypes, automated testing, system integration, as well as deployment. These two activities are depicted in the model as two cycles, which are positioned into a two-dimensional space where the vertical axis is product-market fit and the horizontal one is time. Over time, a successful startup would move from the unknown domain into the known domain in the product-market fit Y-axis as a result of performing these activities. This model could be seen to reflect the two startup learning processes of Blank (2013) discussed in Section 2.1.

Going into more granular detail, Klotins, Unterkalmsteiner & Gorschek (2019) present a customized framework for categorizing startup practices, the

Software and Business Process (SWBP), which they then use to analyze practice use in startups in more detail based on the knowledge areas of the framework. They base the framework on SWEBOOK (Bourque and Fairley, 2014), with the addition of some business aspects based on Osterwalder et al. (2005) and Zachman (2003). Klotins et al. (2019) summarize that while many startups do try to validate their ideas, the practices are "often rudimentary and lack alignment with other knowledge areas." Moreover, they highlight that quality is considered of little importance, occasionally with "disastrous events" as a result.

In another paper, Klotins et al. (2019) discuss anti-patterns in software startup. These are practices, or less specific descriptions of ways of working, patterns, to *avoid*. These include issues such as taking too long to finish an initial version of the product, or an MVP, and lack of customer involvement and lack of validation activities. They break these anti-patterns down into smaller parts in analyzing the issues leading up to these situations in the paper.

To provide a further look into how startups differ from other software organizations in practice, the next subsection looks at the utilization of Agile in the industry through various existing studies and industry surveys. We know that startups prefer Agile practices to what extent they use established practices (Paternoster et al., 2014). However, how Agile practices are actually utilized can vary greatly between organizations, as is discussed next.

### 2.2.2 Agile Development in Startups

Conboy (2009) defines agility in the context of Information Systems Development (ISD) as follows: "the continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment." Indeed, Agile methods are associated with reactivity and the ability to deal with *change*, primarily in relation to software requirements. While this definition does not explicitly contain the notion of iterativeness, the cyclical nature of Agile is nonetheless present in the paper discussing the definition in more detail (Conboy, 2009).

Though the ability to deal with change and reactivity are central to startups struggling with uncertainty, startups are averse to utilizing SE methods, even Agile ones, and only adopt practices later on in the startup process as well (Paternoster et al., 2014). Though startups use Agile practices, they are often used in an ad-hoc fashion (Giardino et al. 2014). This may be a result of Agile methods being well-suited for situations where the problem is understood and the method is there to guide the 'how' part of the work, whereas in the startup context the problem the solution being developed is supposed to address is also unclear (Bosch et al., 2013). This disconnect is also at the root of the Lean Startup methodology of Ries (2011).

Despite decades of research and vast industrial experiences on Agile, Agile is still loosely defined in practice, especially out on the field (Kuhrmann et al., 2021). Kuhrmann et al. (2021), based on a survey of 1467 companies, argue that

under 15% of all companies in fact develop software in a purely Agile or traditional waterfall manner. Moreover, they argue that there are no methods or practices that guarantee or prevent agility. As a result, "being Agile" or "doing Agile" can mean very different things to different individuals and companies.

This is also important to acknowledge when discussing software development in software startups and comparing it to software development in mature organizations. According to the 14th Annual State of Agile report (Digital.ai 2020), a large-scale practitioner survey based on 1121 company responses, up to 95% of software companies utilize Agile development methods. Another recent practitioner survey (GoodFirms Research, 2019) posits that some 84% of companies developing software utilize Agile methods to do so. Less recent academic studies provide somewhat lower estimates (e.g., in 2011, 58% of Finnish companies used Agile or Lean methods according to Rodríguez et al. (2012). Nonetheless, the majority of software companies arguably utilize Agile methods based on both academic and industry surveys.

Yet because 'Agile' is not a strictly defined method (Kuhrmann et al., 2021), these companies can, in practice, work in very different ways and still consider themselves to be using Agile methods. The aforementioned 14th Annual State of Agile report serves to highlight this. The report, among other things, asked the respondents to report which specific Agile practices they utilized. While some practices were very widespread, no practice was utilized by every company who reported being Agile. This is in line with the argument of Kuhrmann et al. (2021) who posit that no specific practice makes a company Agile or not Agile. Companies tailor and mix and match Agile practices to form their own ways of working in the context of Agile. Moreover, companies also routinely tailor Agile methods such as SCRUM into what are referred to as SCRUMbutts ("We use SCRUM, but..."), which results in a situation where a company may report that they are using SCRUM while actually deviating from traditional SCRUM in various ways. In many cases, quality practices are the first ones to be omitted (Ghanbari et al., 2018). Method tailoring is common in the industry in general (Jacobson et al., 2012).

From this follows that, even though some startups utilize Agile methods (Paternoster, 2014), their ways of working can still differ greatly from those of larger companies that also utilize Agile. For example, as many as 85% of software companies utilized daily standup meetings (Digital.ai, 2020), but only 30% of software startups utilized daily standup meetings (Pantiuchina et al., 2017). Given the different survey approaches used to produce these results, they are not directly comparable, but nonetheless give us some insight into the current situation. They underline how heterogeneous the use of Agile can be. While startups also utilize Agile and Agile practices, they seem to prefer different Agile practices than more mature organizations.

In addition to preferring different Agile practices than more mature organizations, startups utilize Agile overall less than mature software organizations. According to Paternoster et al. (2014) "agile and more traditional methodologies struggle to get adopted by startups due to an excessive amount

of uncertainty and high time-pressure" and "software development practices are reported to be adopted only partially and mostly in a late stage of the startup life-cycle". In other words, startups are more likely to mix and match single practices to suit their needs as opposed to using complete methods. Yet even singular Agile practices are notably less common in startups than other types of organizations (Pantiuchina et al., 2017). Thus, while startups may prefer Agile methods compared to other types of methods, the utilization of methods as opposed to various singular practices remains low compared to the rest of the industry, and the way they are utilized also differs compared to mature organizations.

## **2.3 Startup Practices and Methods**

If existing SE methods are poorly utilized by startups (Paternoster et al., 2014), with Agile methods for example better suited for more mature software organizations (Bosch et al., 2013), methods and practices specifically aimed at startups could help. However, as is discussed in this section, such methods and practices are scarce. This section identifies and presents some of the existing methods and practices for startups.

### **2.3.1 Lean Startup**

Arguably the most famous startup-related approach, the Lean Startup methodology is based on the influential book of practitioner expert Eric Ries (2011). In his book, based on his experiences on working on the online chat service Second Life as the CEO of his startup, Ries discusses his lessons learned from that endeavor. Based on these lessons learned, Ries formulates what they refer to as Lean Startup.

The Lean Startup has also since been regularly discussed in academic literature. Aside from being mentioned in passing in a large number of startup-related publications, it is also the focus of a notable number of papers. For example, Bosch et al. (2013) present a model to support the utilization of Lean Startup principles in practice. Similarly, Lean Startup principles have been utilized in Internal Corporate Ventures (ICV), and specifically in Internal Startups, which are a subset of ICV. ICVs are a part of corporate entrepreneurship, alongside strategic entrepreneurship, as discussed by Morris et al. (2010). ICVs are organizations that exist inside existing business organizations and are typically tasked with carrying out innovation in the form of proposing a new product or service for the parent company (Maine, 2008). While poorly defined, internal startups differ from ICVs in that they utilize startup approaches, such as the Lean Startup (Kemell, Risku et al., 2020).

From the point of SE, lean startup can be seen as an approach primarily focused on requirements. It is focused on validating business ideas as well as more specific requirements or assumptions about the idea or solution. Lean startup stresses the importance of learning and using data to both learn and make

informed decisions. In practice, lean startup is not a conventional method in the sense that methods are understood in IS and SE literature, as it does not describe a process. As a result, it is often considered difficult to utilize in practice (Bosch et al., 2013). It is, as such, perhaps closer to what could be described as a ‘philosophy’ or an ‘approach’. Although the lean startup does suggest a collection of tangible practices, these practices do not form a process, and utilizing them can be challenging at times. Various practitioner handbooks and other guidelines to support its adoption and use exist, out of which the book of Maurya (2012) is one of the better-known ones.

The core of lean startup is the so-called Build-Measure-Learn loop. The BML loop is focused on using data for business purposes. The BML loops is about building (“Build”) Minimum Viable Products (MVPs) to collect data (“Measure”) in order to validate assumptions (“Learn”). Each MVP should be built with a plan in mind in terms of the measure and learn stages. MVPs are discussed further in the following subsection.

The lean startup can be considered to present the shift in startup culture that has happened after the early 2000s. As Blank (2013) discusses, during the dot-com bubble, startups typically operated in “stealth mode” while hiding their ideas from their competitors – and, in the process, their potential future customers. When hiding an idea in this fashion, it is difficult to utilize lean startup principles that focus on involving customers and collecting data and feedback. Blank (2013) argues that the lean startup has since rendered this approach focused on secrecy largely obsolete in most cases, although it can still be useful to hide ideas in certain cases.

### **2.3.2 High-Profile Startup Practices**

Some practices associated with startups, and with Lean Startup as well, have become particularly acknowledged out on the field and in academic literature. In particular, the Minimum Viable Product (MVP) and pivoting. In addition, high levels of customer involvement have become associated with software development in startups. These are not entirely novel concepts, however, as customer involvement, for example, is a core principle of Agile development in general. Similarly, MVPs are related to prototypes (e.g., as seen in Nguyen-Duc et al., 2017), although the MVP has become a clearly separate concept in startup literature. A prototype can be an MVP (even if advanced MVPs as initial MVPs can be bad practice (Klotins et al., 2019)), but not all MVPs are prototypes. Nonetheless, such practices have become associated with startups and have been studied in various existing papers (including Article IV), and they are also present in the method presented in Article V.

According to the Lean Startup, every startup should build MVPs in order to gather data that can be used to validate their business idea. The Minimum Viable Product is a key practice in software startups. According to Ries (2011), the types of potential MVPs are numerous. An MVP does not have to be a functional product, or a prototype. An MVP can be, for example, a simple video explaining the product and why a user should buy it, or a seemingly functional



user interface that, under the hood, is not at all what it seems (Nguyen-Duc & Abrahamsson, 2016). Indeed, MVPs are about learning, and there are many ways to gather data about a business idea or an assumption.

Though MVPs are intended to help startups validate their ideas, their use in practice is more multi-faceted. Nguyen-Duc & Abrahamsson (2016) find that MVPs are often reused and retooled to what extent possible, depending on the type of MVP in question. However, MVP use is not necessarily systematic, and few startups seem to produce multiple MVPs of differing types (Nguyen-Duc & Abrahamsson, 2016). When they are used, however, MVPs can produce various benefits:

We found that MVPs could be useful for a startup as a design artifact, a boundary spanning artifact and a reusable artifact. The process from business ideas to a launching product consists not only loops but also parallel branches. When market validation and product design tasks are carried on at the same time, certain types of MVPs would play a role of mutual adjustment between input from customers and product design. (Nguyen-Duc & Abrahamsson, 2016)

Data can help companies make objective decisions. It can be difficult for a CEO to abandon their vision even when data points to it not working, but such difficult decisions may need to be done. The BML loop, in this sense, is but one way of carrying out data-driven decision-making in startups. Moreover, this emphasis on data is also reflected in the other method (or philosophy) discussed in the next section: Growth Hacking.

The BML loop also stresses the importance of using the data to determine whether to persevere (continue with the idea) or to pivot (change direction). In more detail, a pivot can be considered a strategic change designed to test a fundamental hypothesis about a product, business model, or growth engine (Article IV). A pivot can be a fundamental change in the business plan, or it can be any more minor change that changes some aspect of the business model. There are multiple types of pivots. For example, changing the primary platform of your service from iOS to Android would be a platform pivot. The pivot is a well-established practice in startups, discussed by Ries (2011) and in various existing papers (e.g., Bajwa et al., 2016; Bajwa et al., 2017; Bajwa et al., 2020; Khanna et al., 2018; Article IV).

### **2.3.3 Growth Hacking**

Growth Hacking is perhaps currently still predominantly seen in the academia as a “digital marketing buzzword” (Herttua et al., 2017). However, Growth Hacking as a concept has recently been gaining some traction in academic research as well, with papers discussing the concept being published especially in marketing and other business-related research venues (e.g., Bohnsack & Liesner, 2019; Troisi et al., 2020). As is the case with lean startup, Growth Hacking is more focused on business than SE, but still closely involves SE, as is highlighted in this section.

Growth hacking remains loosely defined and calling it a method is not accurate. It is more accurate to consider it a strategy (as Herttua et al. (2017) do) or a model (as Troisi et al. (2020) characterize it). currently considered a *strategy*, and particularly a digital marketing one. Growth hacking is a data-oriented approach to digital marketing and is related to the growing importance of data in business, and especially in startups. Growth hacking is about “hacking growth” to gain new users or customers at a rapid pace, by using low-cost or novel approaches to digital marketing. Its relevance to startup comes from this association with low-cost and novel or innovative practices that can be utilized by startups in particular.

In practice, growth hacking is carried out through the use of growth hacking *techniques*, which could be likened to practices in SE, or practices or techniques in IS literature. These techniques are describe specific ways of acquiring new users. For example, ‘refer-a-friend’ and influencer marketing are considered growth hacking techniques in one practitioner book (Patel & Wormley, 2017). While the goal of Growth Hacking is clear, and the Growth Hacking techniques discussed in grey and black literature are often quite straightforward descriptions of tangible practices, the practices do not form a clear process. GH is about *experimentation*. Not experimentation in the strict scientific sense, but nonetheless experimentation where data is used as a metric of success. GH is about utilizing those GH techniques and finding out what works or does not work for the startup in question. A-B testing and other ways of weighing which techniques produce result and which do not are key in growth hacking.

Whereas academic literature on growth hacking is still scarce, grey literature on growth hacking is common. For example, various books (e.g., Ellis & Brown, 2014; Linkner, 2017; Patel & Wormley, 2017) list singular growth hacking techniques or growth hacking approaches that startups can utilize in an attempt to grow their user base. Grey literature also includes various tools for growth hacking, such as growth hacking funnels, or the Bull’s Eye Framework (Weinberg & Mares, 2014).

The funnel is a business tool that predates growth hacking. The funnel describes the process where, e.g., a website visitor becomes a paying user for a software, going through different stages of the funnel on their way from the start to the end. At every point of the way, some of the initial website visitors are lost, and these conversion rates are important metrics when looking at user (or payer) acquisition and related metrics such as user life-time value. This is why the funnel is shaped like a funnel.

These funnels typically feature five or six stages that describe the conversion process. At the very top of the funnel is often *acquisition*. At the acquisition stage, users first enter the funnel by, e.g., visiting the website of the service. In some funnels, acquisition is preceded by *awareness* where users first become aware of the service but have yet to truly enter the funnel. Following acquisition is *activation* when visitors convert into users by adopting the service, if only to try it out before concluding that it is not for them. When users keep on using the service, they move to the *retention* phase where they become regular

users. If they start paying for the service in the form of, e.g., premium subscriptions, they move to the *revenue* stage of the funnel. Finally, if they are satisfied enough with the product to pay for it and even refer their friends to it, they move to the last *referral* stage. As growth hacking is very focused on data, these funnels provide one source of data to be used while utilizing growth hacking techniques. Underlining, in part, the relevance of the funnel from the point of view of growth hacking, Feiz et al., (2021) utilize the funnel as a framework in their study discussing growth hacking.

While seemingly entirely related to business, the idea of growth hacking is linked with SE. In growth hacking literature, the individual in charge of growth hacking, the so-called growth hacker, is someone also capable of programming. Growth hacking is a form of digital marketing, and often the techniques are technical. Many growth hacking techniques require changes to be made to the software itself, or the software needs to be changed as a result of the lessons learned.

#### **2.3.4 Research-Based Methods, Practices, and Tools for Startups**

Methods for software startups are scarce in general, and especially in academic literature (Unterkalmsteiner et al., 2016). On the other hand, some more specific tools that support specific tasks exist, however. This section discusses some examples of such.

Melegati, Guerra & Wang (2022) propose a technique for eliciting hypotheses based on cognitive mapping, HyMap. It is aimed at early-stage startups looking to clarify their business idea. The hypotheses determined through the use of HyMap can be helpful in helping startups better understand their own ideas in the earlier stages of the process. The authors posit that an assumption is a “personal or team-wise, generally implicit understanding taken as truth without being questioned or proved” whereas a hypothesis is an “explicit statement that has not been proved yet but could be tested through an experiment” (Melegati et al., 2022). I.e., assumptions could be made into hypotheses (by, e.g., using HyMap). The importance of testing assumptions in startups is also discussed by Gutbrod & Münch (2018) as they propose a workshop format for teaching how to prioritize assumptions.

Bosch et al. (2013) present a model for early-stage software startups focused on idea validation. The model comprises three steps: idea generation (generating ideas to work on), the backlog (generated ideas are prioritized into a backlog), and the funnel (ideas are systematically validated). The third step, the funnel, consists of four steps: problem validation, solution validation, MVP validation small-scale, and MVP validation large-scale. The purpose of the model is to provide an actionable framework in which to utilize the Build-Measure-Learn loop of the Lean Startup of Ries (2011).

### 2.3.5 Suitability and Relevance of Existing SE Practices

While the suitability of existing SE methods in the startup context is often questioned, it is arguably not as though they are completely off the mark in that context, especially as far as individual practices are considered. As established so far, though startups seldom use formal methods and even use practices only later on (Paternoster et al., 2014), startups *do* utilize existing Agile practices to some extent (Pantiuchina et al., 2017). Agile *methods* are argued to be ultimately poorly suited for startups. The methods focus on *how* to carry out SE in a context where the *what* is rather well understood, while in the startup context the *what* is the bigger issue (Bosch et al., 2013). On the other hand, individual practices can be better suited for the startup context.

In fact, many of the popular startup practices are ultimately rooted in existing practices. MVPs (Lenarduzzi & Taibi, 2016; Ries, 2011) are closely related to prototypes, although not all MVPs are prototypes. Past the very early stages of a startup's lifecycle, however, prototypes become common as MVPs as product development progresses in the startup. Lean Startup (Ries, 2011) is ultimately also based on Lean, as its name implies, which is a much older philosophy. In a similar fashion, while the Growth Hacking Funnel is at the center of Growth Hacking, marketing funnels date back nearly a century in the form of the *purchase funnel*.

Using prototyping as an example, numerous studies on prototyping both in and out of the startup context exist. The phenomenon is not new, and even *rapid prototyping* has existed long before the MVP paradigm that stresses both speed and learning. Studies on prototyping in startups also discuss the topic and their results in the light of these existing studies (e.g., Nguyen-Duc et al., 2017). On the other hand, Fagerholm et al. (2017) propose a model for continuous experimentation using prototypes, which, while not directly aimed at startups as such, is still validated in a startup context in the paper. While learning in stressed in the startup context in particular, it is at the center of prototyping in general.

Turning to the Lean Startup philosophy, let us look at the Lean principles it is built on. Poppendieck & Poppendieck (2003) discuss seven lean principles:

- Eliminate waste. Waste refers to any anything that does not add value to the software. E.g., coding more features than is needed.
- Amplify learning. Software development should be an iterative learning process.
- Decide as late as possible. Decisions should be based on data rather than assumptions. The best way to avoid uncertainty is building in the option to decide late(r).
- Deliver as fast as possible. The faster you start delivering software, the quicker you start learning from it. Speed is about being able to start receiving feedback from users or customers.
- Empower the team. Local signaling (visible charts, daily meetings...) where the team makes decisions on its own in addition to those coming from above is key.

- Build integrity in. Software should have a coherent architecture, be usable, be suited for its purpose, while also being maintainable, adaptable, and extensible.
- See the whole. A UI designer may be tempted to focus only on refining the UI to be as good as possible, but even specialized experts should still retain a sight of the big picture.

While lean startup does not place direct emphasis on all of these principles, they can still be found in it. Overall, lean startup is about focusing on the essential features (eliminate waste), while aiming to start delivering and learning as quickly as possible. The MVPs simply take the concept of a prototype further and expand on it past conventional, functional prototypes. On the other hand, the lean startup does place less emphasis on building integrity in, as startups are encouraged to fail fast, if failure is imminent.

Indeed, there is arguably plenty of overlap between existing SE practices and methods and those tailored at, or favored by, startups. Though startup research generally reinvents the wheel by arguing that startups are unique, which can be used as an argument to re-investigate any topic in the startup context, the findings of all extant studies may not be as irrelevant for startups as startup literature often argues. Moreover, with startups becoming increasingly common, not all studies utilizing startups as research subjects necessarily frame the study as a startup study in particular (see for example (Fagerholm et al., 2017)).

As such, to conclude this section on startup practices and methods, it should be noted that not all existing practices and methods are necessarily ill-suited for startups. Startups are argued to use various Agile practices despite these practices not being devised to specifically suit the startup context. In fact, it seems to not be entirely clear *why* startups forgo existing methods and practices. It is possible that some practices or methods go unused, for example, because the developers are inexperienced and simply do not know better or do not have the time and will to learn these new processes that do not directly and right away contribute to the tasks at hand.

## 2.4 The Essence Theory of Software Engineering

Proposed by the SEMAT initiative (SEMAT (n.d.)), which consists of both practitioners and academics, The Essence Theory of Software Engineering (Essence from here-on-out) is based on a sizeable endeavour. In practice, Essence is a modelling tool for describing work practices and methods in SE. The official site of Essence summarizes it as a language (Essence in Practice, n.d.).

Essence consists of two main components. The first one is what its authors refer to as a kernel (Jacobson et al., 2012; Object Management Group, 2018), which can be considered a set of building blocks for creating methods and describing practices, as well as a project management tool even without using the language

to extend it further. The second one is the Essence language that is used to describe methods, as well as the Essence kernel itself.

As discussed in the introduction, the original goal of this dissertation was to evaluate the suitability of the Essence kernel for the software startup context, and to devise a method for software startups by using the Essence language to do so. Articles I and II supported this goal. However, based on the results of Article I and the lessons learned in Article III, the method in Article V was ultimately not described using the Essence language. Essence is, however, nonetheless discussed here due to its role in this dissertation in spite of this.

Originally, when planning the research that ultimately produced this dissertation, we were looking for theoretical frameworks for devising a method for software startups. We wanted to use a framework for doing so in order for the method design decisions to be justifiable through the framework. Essence is a well-documented tool for devising methods that has also been studied in academic research.

We considered Essence interesting due to its focus on method tailoring, which is something startups actively engage in. Startups are known to rarely use methods by-the-book and generally prefer various Agile practices over methods (Paternoster et al., 2014), mixing and matching them to suit their own purposes. As startups already engage in ad hoc method tailoring on their own, we felt that Essence could have helped them to do so in a more systematic and planned manner, while also encouraging them to more actively reflect on the way-of-working they chose.

Additionally, we felt that cards are an approachable way of using a method that can be potentially lightweight as well, which would suit startups. However, based on Articles I and III, we concluded that using Essence to describe the method in Article V would have made it too difficult to learn for the card-based approach to be considered lightweight in this context. It is unlikely that startups would be willing to devote resources towards adopting such a method. Though the method was, as such, not described using Essence, the card-based approach itself was considered suitable and was kept, making the method *Essence-inspired* at most. To this end, the methods presented in Articles III and V were still devised using the philosophy behind Essence: the practice of essentializing practices in order to create methods from practices.

The rest of this section further discusses Essence in general. Section 2.4.1 presents the Essence language and its various notational elements. Section 2.4.2 presents the Essence kernel and its contents. Finally, Section 2.4.3 discusses extant research on Essence.

### **2.4.1 The Essence Language**

As the kernel is described using the Essence language, it is in order to discuss the language first. The Essence language provides the means to describe methods with Essence. To this end, it is also how the kernel is extended when using it as a set of building blocks for describing methods.

The language of the specification is to be used to extend the kernel, and it is what ultimately makes the specification modular. The language combines natural and formal languages. It can be utilized on multiple levels of conformance, with varying degrees of formal language used. Lower levels of conformance offer less utility when used on conjunction with external tools, while higher levels of conformance are more easily tracked and used with external tools.

In the Essence language, elements are split into three areas of concern (i.e., categories). These are as follows: (1) customer, (2) solution (i.e., things related to the software system being developed), and (3) endeavour (i.e., team-related things). These areas of concern are color-coded in the Essence language, with the customer area being green, the solution area being yellow, and the endeavour area being blue. This colour-coding is also seen in practice in Figure 6, which depicts the alphas of the Essence kernel. No element belongs into any area of concern by default. An alpha, for example, can belong to any of the three depending on the nature of the alpha, as seen in Figure 6 (Section 2.4.2) as well.

The Essence language contains a number of concepts that are used to describe practices and methods. Some of these elements are also present as building blocks in the kernel. There are eight concepts in the language: alphas, alpha states, activity spaces, activities, competencies, work products, resources, and patterns. These elements of the Essence language are detailed below.

**Alphas and Alpha States** Alphas are, simply put, “things to work with”. These are elements that are central to the project or other endeavour at hand. Alphas are things that are tracked to determine how the work is progressing. Indeed, alpha, in Essence, is an acronym for *Abstract-Level Progress Health Attribute*. As the more specific name behind the acronym implies, alphas are higher-level elements, such as *requirements*, rather than single features of a software. To this end, each alpha has a set of alpha *states*. These alpha states contain progress-related descriptions and criteria that help determine how much progress has been made on the alpha. For example, the requirements alpha progresses from ‘conceived’ where the requirements have only just been formulated, to ‘fulfilled’, where they have been successfully built into the system. (Object Management Group, 2018)

**Activities and Activity Spaces** Activities are “things to do”. These are concrete descriptions of work being carried out. Carrying out activities is what results in progress on alphas and carrying out activities can produce work products. Some competencies may be required to carry out certain activities. Activity spaces can be used to organize activities. (Object Management Group, 2018)

**Competencies** Competencies describe the skills, abilities, or other such qualities required to carry out certain activities in the context of the endeavour. While these can be binary, determining levels of competency with certain skills can be beneficial to better manage a project. (Object Management Group, 2018)

**Work Products** Work products are tangible work outcomes. They can be pieces of software, documents, or any other type of relevant object. Work products are

created by carrying out activities and should be related to an alpha. (Object Management Group, 2018)

**Patterns and Resources** Patterns are “generic concepts that can be attached to any language element.” (Object Management Group, 2018). For example, patterns can be used to sequence activity spaces, or describe roles in a team. Resources, on the other hand, are external resources outside the Essence model being utilized that are referenced from it, such as websites.

#### **2.4.2 The Essence Kernel**

The Essence kernel is a “stripped-down, light-weight set of definitions that captures the essence of effective, scalable software engineering in a practice independent way” (Object Management Group, 2018). It is intended to contain all the key elements that are present in every SE endeavour (Jacobson et al., 2012). In practice, the kernel contains: alphas (and their alpha states), activity spaces, and competencies.

There are seven alphas in the Essence kernel, and they form the core of the specification. These are seen in Figure 6. These alphas serve as a starting point for constructing methods. Each SE project, arguably, has at least these elements in it. The alpha states for these generic kernel alphas are important for keeping track of progress on the project at hand. While constructing method, the users of the specification would then add more alphas to account for relevant, more project-specific concepts to account for.

The alphas form the core of the kernel, but the kernel also includes activity spaces and competencies related to these alphas. These include generic activity spaces such as “Specify the Software” which could contain activities such as “Identify Use Cases” and “Specify Use Cases.” The competencies included in the kernel are core competencies required to carry out such common work tasks.

In practice, this kernel provides building blocks for utilizing the Essence language to construct methods. A method constructed using Essence could use the kernel as a basis, adding alphas, activities and activity spaces, and competencies that are specific to that project, to complement it. These would then be used to formulate practices in order to construct a method. E.g., a practice would be related to an activity (and an alpha) and would require certain competencies. Though the kernel is intended to be extended to create methods, the kernel alone, and as is, can act as a project management tool (Jacobson et al., 2012).



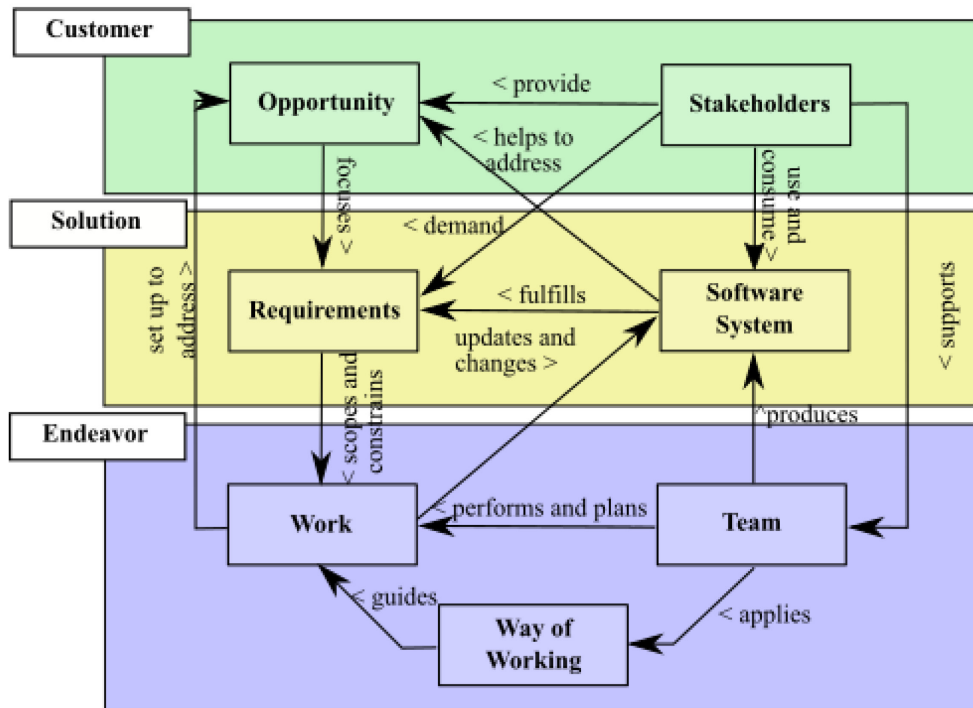


FIGURE 6. Essence kernel alphas (Jacobson et al., 2012; Object Management Group, 2018)

### 2.4.3 Essence in Research

Nearly a decade after its inception, Essence has yet to become widespread out on the field (SEMAT 2018). Similarly, Essence has seen modest interest in academic research thus far. Nonetheless, some academic literature on Essence exists, and I have published papers on Essence not included in this dissertation as well. However, many of the existing studies on Essence are by members of the SEMAT initiative behind Essence.

Existing research suggests that the lack of practitioner interest in Essence may, in part, be a result of its resource-intensive adoption resulting from the specification itself being difficult to understand, as well as a lack of tooling (Graziotin & Abrahamsson 2013) and tutorial resources (Article I). Some tools to help adopt and use Essence have been proposed (e.g., SematAcc (Graziotin & Abrahamsson, 2013), and Essencery (Kemell, Evensen et al., (2019))). However, as most effort in adopting Essence is related to learning to use the language and grasping the idea of the kernel and Essence in general, tools for using can only do so much to facilitate the utilization of the specification. In another study, we develop a board game intended to help its players grasp the idea of Essence in an attempt to make the specification more approachable (Kemell, Risku et al., 2018).

Aside from studies looking to support the use or adoption of Essence through tooling, there are education-focused papers on Essence, papers demonstrating the use of Essence in various ways, as well as empirical papers. Shortly after the Essence specification was published (Jacobson et al., 2012), those

involved with Essence published various papers on it. Park et al. (2016) demonstrate how Scrum can be described with Essence. Ng (2015) uses Essence to support theory-based software engineering in an industry setting. When the specification had only just been published, Ng & Huang (2013) provided preliminary support for the use of Essence in education based on feedback from professors and lecturers. Ng et al. (2013) discuss the potential value of Essence for SE research. Park (2015) provides algorithmic support for utilizing Essence.

Since these early days of Essence, many of the papers on Essence have been carried out in student settings, and many of these papers have focused on education as well. These include Article I, although the goal of Article I was also to evaluate Essence in general, but while using student data to do so. Pieper et al. (2017) discuss applying Essence games in an educational setting. Zmееv & Zmееv (2020) publish on their experiences on having students utilize Essence in a project-based SE course. Overall, it seems that few new SE studies utilizing or focusing on Essence have been published since 2018.

## 3 RESEARCH METHODOLOGY

In this section, I discuss the different methods utilized in the papers included in this dissertation. The first two sections discuss the research methods utilized in the articles of this dissertation. In the first section, I discuss Case Study as a research method (used in Articles II and IV). In the second section, I discuss Action Research as a research method (used in Articles III and V). In the third section, I discuss the data collection and analysis methods used in this dissertation: interviews (Articles II and IV) and thematic analysis (Articles I, II, and IV).

### 3.1 Research Evolution

Originally, this dissertation was to evaluate the Essence Theory of Software Engineering in the context of software startups, and, if needed, to then produce a modified version of Essence (and specifically its kernel) better suited for that context. In practice, this would have entailed the creation of new alphas for the kernel as necessary, as well as activities and activity spaces, and competencies. This would have been in addition to creating a method for early-stage software startups, as is done in Article V. This method would then have been bound to this software startup kernel using the Essence language.

The studies in Article I and II were conducted with this aim in mind. As Essence is predominantly aimed at traditional software organizations, we wanted to first see whether it could be utilized in other organizations as well. While students are not fully analogous to startup, although many startups are founded by students or recent students, we began this endeavor by studying Essence in the context of student projects (Article I). The results indicated that students could grasp the idea of Essence and utilize it. However, it also seemed that it was difficult for them to learn to use Essence. In Article II, we studied the Essence kernel through the lens of software startup practices, evaluating whether additional alphas would be needed to cover the practices utilized by startups.

Some time in early 2019, however, while working on Article III and the ECCOLA method presented in it, we started doubting the suitability of Essence. ECCOLA, like the method in Article V, is a card-based method developed using an action research approach. Whereas the method in Article V is aimed at software startups, ECCOLA is closer to a traditional SE method. While developing ECCOLA, we struggled to make the method practical while adhering to the Essence notation.

In the early stages of the development of ECCOLA, we tested the method with student project teams. We instructed students to read a book on Essence and made sure that they had done so by having them produce a course deliverable on the book. Despite the students having read the book, at least in part, they struggled to grasp the Essence language elements on the cards. Moreover, including the Essence elements brought no benefits. As a result, we opted to incrementally lessen the level of Essence conformance of ECCOLA. These lessons learned carried over to startup method of Article V, changing the aims of this dissertation. As we no longer saw the value of producing a version of the Essence kernel for software startups and binding the method to that kernel, this objective was abandoned. Instead, the focus of the dissertation became the development of the method as a stand-alone method.

While the method, the Startup Cards for Early-Stage Startups, is still card-based and based on the idea of essentializing practices that is the core of Essence, it is no longer conformant with the Essence language to a notable degree. The cards still utilize the colour-coding of the Essence language and are visually similar to Essence cards, but the role of Essence is otherwise small. Though this does not present a notable change in research objectives, it is still documented here to further explain the background of this dissertation and research related to it.

## 3.2 Research Approach

This dissertation took on a qualitative approach to the topic. All of the studies included in this dissertation utilized primarily qualitative data, although, e.g., Article IV included some quantitative analysis of said data through thematic analysis. In practice, this meant utilizing the number of codes to make observations about the data.

Following the research method taxonomy of Järvinen (2001; 2004), this dissertation includes both studies stressing what reality is, as well as studies stressing the utility of artifacts. In terms of reality stressing studies, the dissertation includes both theory-testing (Articles I and II) and theory-creating (Article IV) studies. In terms of artifact-related studies, Articles III and V are both artifact-building and artifact-evaluating studies. As Järvinen (2004) notes, an action research approach includes both artifact-building and evaluation. However, this classification of Articles III and V assumes that *artifact* includes SE methods, which is more akin to how design science understands the concept of

artifact, as opposed to classifications that only include more traditional IT artifacts such as software systems.

Though, according to some, qualitative studies can sometimes struggle with generalizability due to the low number of cases or low amount of data otherwise, the studies included in this dissertation generally contained satisfactory amounts of empirical data. The data used in each paper is detailed in Table 1 below, along with the research approach and methods of each study.

TABLE 1. Research approach overview of the included articles

Article	Research Approach	Objective(s)	Research Methods
I	Theory-testing	Test Essence in a classroom / student project setting to evaluate suitability for startup context as well.	Case study (102 student teams).
II	Theory-testing	Test the suitability of Essence in a software startup context. Produce list of startup practices.	Case study (13 startups). Qualitative interviews as data.
III	Artifact-building & Artifact-evaluating	Develop card-based software engineering method for AI ethics.	Action research (various types of organizations, incl. 27 student teams).
IV	Theory-creating	Study decision-making in software startup context. Create typology of startups based on decision-making logics.	Case study (40 startups). Qualitative interviews as data.
V	Artifact-building & Artifact-evaluating	Develop card-based software engineering method for startups.	Action research (43 startups).

Article I is a qualitative study. It features a study of 102 student teams utilizing the Essence Theory of Software Engineering. This is a theory-testing study where Essence is evaluated in a student setting. As data, we utilized course reports written by the teams where they, among other things, discussed their use of Essence during the project.

Article II is another qualitative, theory-testing study where Essence is evaluated. In Article II, Essence is evaluated through startup practices. The study itself focuses on startup practices, using empirical data and an existing list of practices to compile a list of common startup practices. These practices are then inserted into the framework of the Essence alphas to evaluate whether these alphas fit the alphas of the Essence kernel as is, or whether additional alphas are required to make Essence more suitable for the startup context.

Article IV is a qualitative, theory-creating study. Using interview data from 40 startups, we conduct a thematic analysis of the data in order to understand decision-making in relation to SE decisions. As a result, we propose a taxonomy of software startups based on decision-making logics.

Articles III and V are similar methodologically. Both articles utilize an AR approach to develop and test an artifact (method) in a practical setting. The methods, in terms of how they are presented, are also similar. Both methods are card-based, the layout of the cards in both methods is very similar, and both methods were originally going to be described using Essence. Because of this, lessons learned could be shared between these two studies taking place simultaneously between 2019 and 2021. In terms of content, however, the methods are different: Article III presents a method for implementing Artificial Intelligence (AI) ethics in practice, and Article V presents a method for early-stage software startups.

In the following subsections of this section, I describe the utilized research methods and data collection and analysis methods in more detail. While doing so, I also discuss in more detail how they were used in the articles of this dissertation.

### **3.3 Case Study**

Case studies are common qualitative research methods. In IS, the qualitative case study has historically been the most common qualitative method (Orlikowski and Baroudi, 1991; Alavi and Carlson, 1992). According to Yin (2002) case study as a method can be defined as follows: “a case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.” Case studies are best suited for answering ‘how’ and ‘why’ questions in terms of research questions (Yin, 2002).

When discussing case study research as a method, a ‘case’ is the unit of study (Myers, 1997). In the context of organizational research, a case is typically a company. Case study, as a research method, can be utilized in conjunction with various data collection and analysis methods. Qualitative interviews are commonly used to form the bulk of the data used in case study research. They can be supplemented with various documents (emails etc.) or observation. Case study research takes no stance on the underlying etymological philosophy, although some recommend specific etymological approaches to case study research (e.g., Yin 2002 is a proponent of positivist case studies).

Case studies vary greatly in depth. A case study may consist of a single interview with one respondent for each company. Some case studies, on the other hand, may comprise multiple interviews with multiple respondents per case company, with the interviews sometimes conducted with years in-between to gain a longitudinal understanding of some phenomenon concerning the case company. Some case studies only feature one case company. These are referred to as single case studies, whereas case studies with multiple cases are, fittingly, considered multiple case studies.

In this dissertation, I utilized case study research in Articles II and IV. In both studies, semi-structured interviews were the primary empirical data used.

In addition, we utilized some observation data and document data as supporting data in Article IV.

In Article II we had 13 software startup cases. Data from these cases were collected through thematic interviews. Most cases had one respondent who was interviewed once, while one case had two respondents and one case had three respondents who were interviewed once. All case startups included in the study had at least three employees and software played a role in their core business value.

In Article IV we contacted 306 startups to ask them to participate in the study. After excluding startups that did not meet our selection criteria, or were not interested in participating, we were left with 40 case startups. To gather suitable data, we utilized the following five selection criteria to select our case startups: (1) the startup has at least two full-time members, so that their MVP development is not individual activities, (2) the startup has operated for at least six months, so that they have had time to accrue relevant experience, (3) the startup has at least a first running prototype, so that their prototyping practices can be discussed, (4) the startup that has at least an initial customer set, i.e., first customer payments or a group of users, so that it has reached some milestones of progress, and (5) software is a central part of the core business value of the startup.

### **3.4 Action Research**

Action Research dates back to the Second World War. It has its roots in practice, having been as a way of conducting social research in practical settings. Specifically, AR was initially used to study extremism and radical thought such as anti-Semitism and right-wing extremism in real-world settings. Rather than focusing on individuals from a psychological point of view, AR looked at groups of people and group dynamics, i.e., organizations. Since then, AR has become a method commonly seen in organizational research in general, as well as IS.

In brief, modern AR is focused on solving organizational problems. In IS, AR has been seen as a way conducting practical research that would benefit practitioners out on the field. AR is participatory in that the researcher and the organization that is the unit of the study collaborate in various ways in order to address the problems the organization is facing (or the problem the researcher wants to solve).

The AR tradition in IS largely draws on the influential paper of Susman and Evered (1978), where they argue for the relevance and rigor of AR. Susman and Evered (1978) consider AR a cyclical or iterative process with distinct phases. These iterations vary in length and may not always include all the phases, however, depending on the research context at hand. The cyclical model for AR proposed by Susman & Evered (1978) is illustrated below. IS scholars have built on the cyclical model of Susman & Evered (1978) to make it, they argue, better suited for IS research. In particular, Davison et al. (2004) discuss what is referred

to as Canonical Action Research in IS, and which has become the common choice for AR in IS.

In this dissertation, I utilized AR in two of the papers included: Article III and Article V (which are discussed in more detail in the fourth section of this dissertation). In Article III we present a method for AI ethics, and the paper was later on expanded upon in Vakkuri et al. (2021), where the method description is more robust given that it is a journal publication, allowing for more pages. The method is developed using AR. In Article V, we develop a software startup method using AR.

In both papers, we utilize the Action Research model of Susman and Evered (1978) as a base while supplementing it with the additions of Davison et al. (2004) and their Canonical Action Research. These two approaches are not mutually exclusive. Rather, Davison et al. (2004) add more detail and guidelines to the original AR process described by Susman and Evered (1978).

In both articles, the method proposed in the article is developed over multiple AR cycles. In Article III, we develop the ECCOLA method over five AR cycles and two years. In Article V, we develop the Startup Cards over four AR cycles and four years. Both methods are deployed and tested in practical settings multiple times over these AR cycles.

### **3.5 Data Collection and Analysis Methods**

The first subsection of this section discusses qualitative interviews, which were utilized in multiple articles included in this dissertation. The second subsection discusses thematic analysis, which was used to analyze data in three of the five articles.

#### **3.5.1 Qualitative Interview**

The interview is the most common data collection method in qualitative research (Myers & Newman, 2007). It is used in qualitative research across disciplines, including SE and IS. Interviews are commonly used in organizational research, given that organizations are entirely reliant on the humans that comprise them. For example, studies focusing on project management often carry out interviews with project managers and upper management. The interactive nature of the interview makes it possible to ask case-specific questions from a respondent, or to direct the interview into new directions if considered beneficial or necessary (Alshenqeeti, 2014), unless conducting a structured interview – rare as they are.

Interview data can be qualitative or quantitative, although quantitative interviews are rarely used. In most cases interviews are used to collect qualitative data. Qualitative interview data can be collected using structured, semi-structured, or unstructured interviews. In a structured interview, a specific set of questions is prepared beforehand, and these same questions, word for word, are asked from each respondent. This set of questions are referred to as the interview



instrument. In a semi-structured interview, the interview instrument is not exhaustive. The interview instrument of a semi-structured interview is more akin to a guideline that directs the interview but does not dictate it entirely. This leaves room for further questions that can be used to collect additional data from the respondents. For example, if a particular respondent is very knowledgeable about one theme in the interview, this theme can be discussed in more depth with that particular respondent. Finally, it is debatable if unstructured interviews exist at all, or whether they are simply less structured semi-structured interviews. An unstructured interview would be an interview without an interview instrument or pre-determined questions, but arguably a researcher always has some questions or themes in mind before an interview. As such, it could be said, rather, that semi-structured interviews can vary in how structured they are.

Data from qualitative interviews is utilized in two papers in this dissertation. Qualitative interviews were used to collect all the empirical data in Articles II and IV. In both cases, the interviews were used as a part of a multiple case study of software startups.

In Article II, we interviewed 13 software startups. The interviews were carried out as semi-structured interviews, using a thematic interview approach. With technical respondents, we utilized an interview instrument with more technical questions related to software development practices. With less technical respondents, such as founders with exclusively a business background, we utilized an interview instrument built around the alphas of the Essence Theory of Software Engineering (see Section 2.5).

As the aim of the study was to validate an existing list of startup practices (from Dande et al., 2014) and to supplement it with new ones if possible, utilizing two interview instruments contributed to a better triangulation of data, as suggested by Langley (1999) in the context of process data. In this case, focusing solely on technical SE aspects could have omitted some less technical practices, and vice versa.

After the interviews, the data was transcribed, and the transcripts were used for data analysis. The data was analysed using thematic analysis as discussed by Cruzes and Dybå (2011). Thematic analysis, also used in Article I and Article IV to analyse data, is discussed in following subsection.

For Article IV, we contacted 306 startups for interviews. After excluding startups that did not meet our selection criteria, or were not interested in participating, we collected data from 40 software startups. To gather suitable data, we utilized the following five selection criteria to select our case startups: (1) the startup has at least two full-time members, so that their MVP development is not individual activities, (2) the startup has operated for at least six months, so that they have had time to accrue relevant experience, (3) the startup has at least a first running prototype, so that their prototyping practices can be discussed, (4) the startup that has at least an initial customer set, i.e., first customer payments or a group of users, so that it has reached some milestones of progress, and (5) software is a central part of the core business value of the startup.

Before setting up the interviews, we designed an interview instrument for the semi-structured interviews. The one used in Article IV was modified from an existing interview instrument used to gather data on pivots in software startup. This modified instrument was piloted with startups and adjusted as needed before being utilized to collect the data for the study. After finalizing the interview instrument, it contained a number of pre-planned, guiding questions for each interview. Past these pre-planned questions, the semi-structured interviews would proceed in a case-specific manner based on the answers.

Subsequently, the interviews were transcribed by a third party organization based on the recordings. The resulting 313 pages of transcripts were used for data analysis. The data was analyzed using thematic analysis, which we discuss in the next section of this section.

Typically, when interviews are used in organizational research, the selection of the respondents warrants consideration. In some cases, the researchers may have to select their respondents based on their point of access (i.e., how, or rather, through whom they enter the organization) and whom it lets them interview. For example, in large, multinational corporations, getting to interview the CEO may be impossible without personal contacts. On the other hand, the CEO may not always be the best person to answer questions in the first place, depending on the topic being studied. A CEO might seem like the person with the most knowledge about the company overall, but in the case of larger organizations, various experts may be better suited to answer more specific questions about the company, or e.g., questions about one specific project.

In the case of startup research, as is the case in this dissertation, these issues are often not as prevalent. Startups typically exhibit flat organizational structures and often have small teams as well (Paternoster et al. 2014). This often makes it easier to set up interviews with founders, as well as CEOs and other key personnel in practice. This also means that CEOs and founders have more in-depth knowledge about the entire company on a ground level. In Article II for example, all the interviews were conducted with CEOs or founders. However, the choice of a respondent can still be important, as a non-technical founder may not be able to answer technical questions about SE in the company.

### **3.5.2 Thematic Analysis**

Thematic analysis is a common method for data analysis in empirical SE and is “used for identifying, analyzing, and reporting patterns (themes) within data in primary qualitative research” (Cruzes and Dybå, 2011). Thematic analysis is common for analyzing qualitative data across fields of science. Braun & Clarke (2006) propose six steps for carrying out thematic analysis:

1. familiarizing with data,
2. generating initial codes,
3. searching for themes,
4. reviewing themes,
5. defining and naming themes, and

## 6. producing the report.

This process is the standard for thematic analysis. Thematic analysis may or may not utilize an existing framework for devising the codes and/or themes (Cruzes & Dybå 2011).

Thematic analysis was used to analyze data in three papers in this dissertation: Articles I, II, and IV. In Article I it was used to analyze data from 102 course reports written by student teams during a practical SE project course, one report per team. In Article II it was used to analyze thematic interview data collected from 13 software startups. In Article IV it was used to analyze semi-structured interview data from 40 software startups. All three articles followed the six-step-process of Braun & Clarke (2006).

In Article I, thematic analysis was utilized due to large volume of data, and because we had no pre-conception of how the students may have felt about using Essence. In the 102 reports analyzed for the paper, the parts relating to the use of Essence were analyzed. While most of the report contents were other course deliverables, each report contained a section on the use of Essence. This section, while otherwise freeform, was to describe: (1) what they thought was good about Essence, (2) what they thought was bad about Essence, and (3) how they had utilized Essence during their course project.

We utilized an inductive approach where the codes arose from the data. These reports were initially read while making notes and saving quotes in a separate document. At the same time, the initial codes were formulated iteratively based on the report contents. This process was then iterative as the codes became final, with older, already read reports re-read and updated with new codes. Once all the reports had been coded and no more codes were considered relevant, the codes were arranged into four high-level themes: difficult or resource-intensive to learn (Essence), inexperience (of the team), way-of-working and method prison, and progress control. These themes captured the main findings of the study and were used to structure the reporting of the findings in Article I.

In Article II, we utilized thematic analysis to analyze interview data. As the goal was to uncover novel practices, in addition to validating existing ones listed in Dande et al. (2014), we utilized an inductive approach to support the elicitation of new practices. First, the data was read in its entirety to gain an overview of the data, and to begin formulating codes. Then, the interview transcripts were coded one by one. Finally, the codes were arranged into themes that were relevant across interviews. In Article II, the purpose of the thematic analysis was to uncover new practices while providing context for the practices (with e.g., codes such as prototype). Practices discussed by two or more case startups were considered prevalent enough to be included into the list of practices presented in this paper.

In Article IV, we used thematic analysis to be able to produce a model with higher-order themes describing some facets of how software startups develop software. We utilized two coding systems. First, we developed a coding scheme for SE activities by using the Software Engineering Body of Knowledge,

SWEBOK (Bourque and Fairley, 2014). This resulted in seven codes: P0. SE (general), P1. Requirement Engineering, P2. Product Design, P3. Software Construction, P4. Software Testing, P5. Software Maintenance, and P6. Software Process Management. Secondly, we developed a coding scheme for effectuation-driven and causation-driven business logics. There were 4 code categories and 18 codes in total for effectuation, and 4 code categories and 17 codes in total for causation. These codes were generated by applying a descriptive coding technique to identify entrepreneurial logic dimensions across the cases, as suggested by Rueson & Höst (2009).

While analyzing the data we looked at these two types of codes in conjunction. We were interested in seeing how these two types of business logics were applied to SE decisions. As such, in addition to looking at decisions made, we utilized the two coding schemes to analyze decision-making related to different types of SE events. In Article IV, we also utilized a mixed research approach where we quantitatively analyzed the qualitative data, placing much emphasis how many times each code appeared (and in conjunction with which codes etc.) to propose the startup typology based on decision-making presented in the paper.

Overall, inductive approaches were used in these studies due to software startup research still being a relatively new research area (Unterkalmsteiner et al., 2016; Klotins et al. 2018). Inductive research where conclusions are drawn through a bottom-up exploration of evidence (data) is well-suited for such research and is a common approach in empirical SE (Seaman, 1999; Wohlin and Aurum, 2015; Ayala et al., 2018; Khurum et al., 2015).

## 4 OVERVIEW OF THE ARTICLES

In this section, each article included in this dissertation is summarized. There are five subsections, one for each article. For each article, the subsection includes a description of the research objectives of the study, the findings of the study, and an explanation of how the article in question is connected to the dissertation.

### 4.1 Article I: The Essence Theory of Software Engineering - Large-Scale Classroom Experiences from 450+ Software Engineering BSc Students

Kemell, K.-K., Nguyen-Duc, A., Wang, X., Risku, J., & Abrahamsson, P. (2018). The essence theory of software engineering: large-scale classroom experiences from 450+ software engineering BSc students. In *Product-Focused Software Process Improvement, PROFES 2018* (pp. 123-138). Lecture Notes in Computer Science, 11271. Springer, Cham.

#### **Research Objectives**

The Essence Theory of Software Engineering is intended to provide any organization with tools to create their own ways-of-working (methods), as method tailoring is common practice in the industry. However, out on the field, Essence is primarily utilized by large, mature software organizations. There is (or was at the time) little research on using Essence in a startup or student setting. The goal of this paper was to understand whether students could utilize Essence in an educational project setting and learn something about method tailoring in the process. As method tailoring is common in the industry and following methods by the book is less common, understanding method tailoring and tools used to do so corresponds to industry needs. The research objectives of the study were summarized into two research questions: (1) How useful do bachelor level students find Essence? (2) What are the challenges in adopting Essence,

specifically for inexperienced software developers, and what could be done to make its adoption easier?

## Findings

The key finding of this paper was that student teams were successfully able to utilize Essence, even though Essence is aimed at established software organization. In more detail, we summarized our findings from this study as follows:

- Essence can teach students new methods and practices by encouraging them to study them in order to tailor their own methods using Essence.
- Essence encourages students to adjust their way of working based on the SE context at hand as opposed to following existing methods by the book.
- Essence helps students structure their way of working in a practical setting.
- Essence is difficult to learn. Better tutorial resources for Essence are needed to make it easier to adopt.

## Connection to the Dissertation

Originally, the study in Article I was conducted to see if and how Essence would work outside its intended context of large industry organizations. Essence has not seen widespread industry adoption (SEMAT, 2018) and the use of Essence is typically looked at in large or multinational software organizations. We wanted to see if student teams could *independently* utilize Essence. More specifically, the purpose was to see whether this was possible in a student context *before* using Essence in a startup context, as a part the original plans for Article V, where the method of Article V was to be described using Essence. As the students were able to utilize Essence successfully to some extent, this paper, in 2018, served as a motivation to continue the originally planned research where Essence played a larger role.

However, while the results demonstrated that students could successfully utilize Essence to some extent, one of the key findings of the paper was also that Essence was difficult to learn. Later, together with the lessons learned in Article III, this contributed to the method presented in Article *not* being described using the Essence language.

Additionally, the use of students in Article I also warrants some discussion from the point of view of this dissertation. As mentioned, the purpose of Article I was to evaluate the suitability of Essence outside the context of large industry organizations first and foremost. For this purpose, the use of student teams is arguably justified, as they provide an environment that is certainly different from that of large industry organizations. Additionally, I argue that the student teams in Article I are reasonably *startup-like environments*.

A SE student team working on a practical SE project shares many of the characteristics Paternoster et al. (2014) associate with software startups. Namely, student teams also exhibit the following characteristics: small team, one product, flat organization, low-experienced team, and little working history. These are characteristics *commonly* associated with startups, and as such, much like how a student may have extensive industry work experience, so can a startup developer,

but on average they would seem to apply to both. Time pressure also applies to both, but for the student team it is related to course deadlines rather than, for example, company financials.

In this light, I argue that Essence being difficult to learn for student teams is also relevant from the point of view of startups as well. If students, who were instructed to study Essence as a part of their studies, found Essence difficult to learn, it is unlikely that startups battling time pressure and a lack of resources would be willing to devote resources towards doing so. This finding already made us decide on a low level of Essence conformance for the initial version of the method of Article V in 2018. Higher levels of Essence conformance were attempted in Article III, on the other hand, which cemented my decision to lessen the role of Essence in the method of Article V.

## **4.2 Article II: Software Startup Practices – Software Development in Startups Through the Lens of the Essence Theory of Software Engineering**

Kemell, K.-K., Ravaska, V., Nguyen-Duc, A., & Abrahamsson, P. (2020). Software startup practices – software development in startups through the lens of the Essence theory of software engineering. In PROFES 2020: 21st International Conference on Product-Focused Software Process Improvement, Proceedings (pp. 402-418). Springer. Lecture Notes in Computer Science, 12562.

### **Research Objectives**

The philosophy behind the Essence Theory of Software Engineering is that methods consist of practices. In order to model methods, one should first describe practices. There are various practices already described in the Essence language that can be found in the Essence Practice Library (Ivar Jacobson International, n.d.). However, these existing practices are not focused on startups. To facilitate the utilization of Essence in the startup context, and to better understand software development in software startups, this study looked at the practice use in software startups. More specifically, the objectives of this study were to (1) find out what practices are commonly used by software startups and (2) to study how the seven alphas of Essence fit the context of software startups, and whether additional alphas would be needed to accommodate common software startup practices. By producing a list of practices commonly used by software startups, we wanted to create a list of practices that could then be utilized to create methods for software startups with Essence.

### **Findings**

We built on an existing list published as a work product from a Finnish project, using empirical data from qualitative interviews to validate said list while adding new practices as they were uncovered. Building on the list, we propose 76 startup practices that can be used to build methods using Essence. Additionally, we

argue that the business aspect is more closely intertwined with SE in startups than other types of organizations. As such, when utilizing Essence for startups, including new alphas related to business could be beneficial. Such new alphas could be, for example, business model, funding, and marketing. These should be under a new, fourth area of concern: business. However, these changes are not formally carried out in the paper and remain an open future research direction – or a practical implication for the users of Essence.

### **Connection to the Dissertation**

Article II provides insights into how software startups develop software by studying individual work practices. The aim of the dissertation was, originally, to study Essence in the startup context and to ultimately produce a version of the Essence kernel tailored for software startups. However, along the way, Essence became an obstacle in the process. Article I already highlights the difficulty of adopting Essence, and this is further seen in Article III, as is discussed next. Instead of simply learning to use the new method, if it is described in Essence, its users will first have to learn to use Essence, which, as discussed in Article I, is not simple, especially for novice developers such as startup practitioners. Nonetheless, though the method in Article V is not a version of Essence, nor formally described using the Essence language anymore, this paper provides building blocks for using Essence in the startup context. Practices, in Essence, are used to describe methods, and so, by building a list of startup practices, we produced a list of building blocks for method engineering in this context.

### **4.3 Article III: ECCOLA - A Method for Implementing Ethically Aligned AI Systems**

Vakkuri, V., Kemell, K. -K., & Abrahamsson, P. (2020). ECCOLA - a method for implementing ethically aligned AI systems. In Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, pp. 195-204, doi: 10.1109/SEAA51224.2020.00043.

#### **Research Objectives**

The goal of the paper was to develop a method for implementing AI ethics. AI ethics is a very topical field following recent process on AI. However, the field has been active primarily in terms of theoretical discussion focused on defining AI ethics through various principles. Empirical studies, on the other hand, are lacking. To remedy this situation, we conducted a series of empirical studies in order to understand the current state of the field (Vakkuri, Kemell & Abrahamsson, 2019; Vakkuri et al., 2020; Vakkuri et al., 2022). Based on these studies, we concluded that there was a need for further tooling to bridge the evident gap between research and practice in the area. To do so, we set out to devise a method for implementing AI Ethics: ECCOLA. In this paper, we



presented the first published version of the ECCOLA method, which was developed by using Action Research.

## Findings

The paper proposes a method, ECCOLA, for implementing ethics in AI. The method was developed iteratively by utilizing Action Research (AR). Over the multiple AR cycles, the method was improved based on the data collected. The paper presents the first public version of the method (found online on Figshare via Vakkuri, Kemell & Abrahamsson (2020)). The method presents a notable contribution in the field of AI ethics, which had been characterized by a lack of actionable tools to implement ethics in practice.

## Connection to the Dissertation

Article III presents an exercise in method development. Its role in this dissertation is its relevance to Article V. The lessons learned from Article III contributed to Article V in various ways – and, in fact, vice versa. These two method development endeavors ran in tandem between 2018 and 2021 and lessons learned from one often influenced the other early on.

These shared lessons learned were possible due to the similarities the two endeavors shared. Specifically, (1) both methods are *card-based* method and share various high-level design decisions, (2) both methods were developed through a cyclical AR process, and (3) both methods were originally intended to be described using the Essence language. While the contents of the cards were different, with one method being an AI Ethics method (Article III) and one being a software startup method (Article V), lessons learned could be utilized in relation to the card layout and how the cards are utilized in practice. As both methods were developed using an iterative AR approach, the methods were changed iteratively based on these lessons learned.

Originally, we planned on describing both methods using the Essence language. Though the results of Article I already indicated that Essence was likely to be difficult to adopt, this alone did not deter us from this decision. The earlier versions of ECCOLA (Article III) were still described using Essence. However, as we tested these initial versions of ECCOLA, we noticed that the teams using ECCOLA were having issues with the Essence elements on the cards. The teams were having issues utilizing ECCOLA because they *also* had to learn Essence to do so, and the teams that struggled to grasp Essence were having difficulties using the cards. We felt that this added needless complexity to the method adoption process.

Consequently, we began to lessen the role of Essence in ECCOLA incrementally, and ultimately it became minimal. This was a decision that carried over to the method of Article V. Nonetheless, ECCOLA and the method in Article V were ultimately still devised using the philosophy behind Essence: the practice of essentializing practices in order to create methods from practices.

Aside from lessons learned related to Essence, the joint method development processes of Articles III and V contributed to each other in terms of the card layout and the overall design of the method. In the end, the cards of both

ECCOLA and the Startup Cards of Article V share a similar layout. The textual contents of the cards of both methods are split into three categories, two of which are directly shared between methods ('motivation' and 'what to do'). Moreover, both methods share various high-level design decisions: (1) both methods are modular (i.e., a subset of the cards can be selected on a case-by-case basis), (2) both methods work in conjunction with existing SE methods, (3) both methods support iterative development, and (4) being modular, neither method proposes a strict process that has to be followed each time, iteratively or not.

In this fashion, Articles III and V are ultimately closely linked together. Lessons learned from one contributed to the other between 2018 and 2021. Article III was later expanded on, and the extended version was published in *Journal of Systems and Software* (see Vakkuri, Kemell et. al (2021)).

#### **4.4 Article IV: The Entrepreneurial Logic of Startup Software Development - A Study of 40 Software Startups**

Nguyen-Duc, A., Kemell, KK. & Abrahamsson, P. (2021). The entrepreneurial logic of startup software development: a study of 40 software startups. *Empirical Software Engineering*, 26(91).

##### **Research Objectives**

Startups are characterized, among other factors, by the strong influence key personnel hold over the companies. Due to the small team sizes, CEOs and CTOs exert particularly large influence over startups. As a result, much of the decision-making in startups, as well as their success, hinges on these entrepreneurial personalities. Though extant studies in business disciplines recognize the influence of entrepreneurial characteristics over companies, including startups, it has scarcely been studied in SE. In this paper, we studied how SE-related decisions are made in software startups. In doing so, we utilized two entrepreneurial logic theories from entrepreneurship literature – effectuation and causation – to categorize decisions made in software startups. More specifically, the research objectives of the study were summarized in two research questions: (1) How do entrepreneurial logics apply to SE activities in startups? (2) How do entrepreneurial logics apply to software product development at the company level?

##### **Findings**

The primary finding of the study was the typology for startups that is based on decision-making characteristics. We proposed that startups could be classified into Type One (effectuation-dominant), Type Two (mixed logic), or Type Three (causation-dominant) startups. By classifying startups into these categories, it is easier to reason about their SE decision-making by acknowledging the characteristics of each type. Startups can, however, change category as they operate, and as such a Type One startup is not guaranteed to stay Type One

indefinitely. As founders and CEOs exert notable influence over startups, it is possible that shifts in their decision-making tendencies also reflect in their startups.

### **Connection to the Dissertation**

SE-related decision-making is closely related to method and practice use. Understanding how SE decisions are made in startups helps us understand why startups choose to work in certain ways. However, our findings regarding decision-making in this study alone could not yet provide ways of utilizing this three-point typology for software startups to influence method and practice related decisions in startups.

On the other hand, in terms of this dissertation, the most interesting findings from this paper were the following observations: (1) Many startups are not successful in learning from their MVPs (due to the effectuation-driven behaviour). MVPs are not used for learning purposes and are reused for different purposes and in different scenarios, and (2) Testing can be minimalistic and effectuation-driven unless the core value proposition is quality (e.g., in the form of safety in safety-critical areas such as healthcare).

These findings were used to guide the creation of the method. First, stressing the importance of learning is important in a method for software startups. We have done this in the Startup Cards method. Secondly, though quality cannot be forgotten entirely, it is not a foremost priority for most software startups, which is also reflected in the method.

## **4.5 Article V: Startup Cards - A Method for Early-Stage Software Startups**

Kemell, K.-K., Nguyen-Duc, A, Suoranta, M. & Abrahamsson, P. (2022). A card-based method for early-stage software startups. Submitted to a Journal for Review.

### **Research Objectives**

Startups struggle to utilize existing SE methods. These methods are aimed at mature software organizations and fail to account for the unique context of software startups. In the absence of methods suited for startups, startups utilize singular Agile practices or simply develop software ad hoc. To remedy this situation, we propose a method for early-stage software startups that contains key software startup practices. This method, the Startup Cards, is created with key software startup challenges and anti-patterns in mind in order to help startups remedy these issues. The method is developed using Action Research (AR), over the course of four Action Research cycles where it is utilized in a practical setting by startup teams. More specifically, the method is developed in an action-based entrepreneurship course (Rasmussen & Sørheim, 2006), or a learning "through" entrepreneurship course (Sirelkhatim & Gangi, 2015) where student

teams work on real startups as startup teams. Based on data from its use, the method is iteratively improved during the AR process.

### **Findings**

Article V presents a method, the Startup Cards, for early-stage software startups. The method takes on the form of a deck of 17 cards, with each card containing one practice. The Startup Cards are intended to help startups tackle key startup challenges. The cards describe key software startup practices that are based on academic and practitioner literature. The method has been iteratively developed over the course of four years, with improvements made based on data from its utilization by startup teams. Article V presents the first published version of the method.

### **Connection to the Dissertation**

This article presents the main contribution of this dissertation. The main objective of this dissertation was to produce a method for early-stage software startups. That method is presented in Article V.

## 5 RESULTS AND CONTRIBUTIONS

This section presents the results and contributions of this dissertation, as well as the threats to their validity and their limitations. In Section 5.1, the results of the articles and thus the dissertation are summarized and further split into knowledge areas in the process. In Section 5.2, the threats to validity of the studies of this dissertation are discussed. These threats are approached through the four aspects of validity discussed by Runeson & Höst (2009): construct validity, internal validity, external validity, and reliability. Then, in Section 5.3, I discuss the practical and theoretical implications of the results (contributions). Section 5.4 concludes this section with a discussion on the limitations of the dissertation.

### 5.1 Results

This subsection discusses the results of this dissertation. While discussing the results, some observations and contributions are highlighted as Primary Empirical Contributions (PECs). These are numbered. Aside from providing concise summaries of the results of the articles, the PECs then act as a framework for the contributions section (Section 5.3). In Section 5.3, their implications are discussed in relation practice and existing scientific literature.

#### 5.1.1 The Essence Theory of Software Engineering in a Student and Startup Context

Though ultimately the method develop in Article V was not described formally using Essence, many of the results of this dissertation are nonetheless related to Essence. Articles I and II in particular present results related to Essence, among other results. Similarly, though the Startup Cards presented in Article V are not formally described using Essence, utilizing them with Essence can be done with some extra effort in binding them to the Essence kernel.

Article I studied Essence in the context of a large number (n=102) of student projects, with each project having a team of students working on it. At the time of its publication, it was one of the few Essence studies focused on an educational context using empirical data. Since then, a few more Essence papers focused on education have been published. While Essence is first and foremost a tool for software companies that lets them describe methods and practices using the Essence language, and to keep track of progress on the project by using the Essence kernel alongside any custom-tailored Essence alphas, Essence can also be useful in teaching students the idea of method tailoring. If, as the authors of Essence argue (Jacobson et al., 2012; Jacobson & Stimson, 2018) one problem out on the field is that organizations are stuck in so-called method prisons, i.e., stuck using methods unsuited for the current context simply because that is the way they are used to doing things, educating future professionals to avoid this type of thinking may be helpful as a long-term remedy.

However, based on Article I, it also seems Essence is difficult to adopt, especially for novice software developers such as students. Essence is a heavy tool that necessitates learning the language *and* the kernel, and as a result is more complex than a pure modelling language such as UML. Moreover, there are (or were at the time) few tutorial resources available. If reading a book 300+ pages long is the only way to learn to use the tool, its adoption can become a daunting task. Past this difficult adoption, though, it seems that Essence can be helpful even for more novice software developers. The kernel helps keep track of progress on the project, while the language can be used to better grasp the method being used in order to make changes to it. In our study, we look at students mainly using SCRUM, and utilizing Essence to describe SCRUM gives them a way of modifying SCRUM to better suit their project context. In this fashion, Essence can encourage critical thinking when it comes to method use.

From this I derive the first two PECs of this dissertation:

**PEC1** Students may be taught the idea of method tailoring through Essence. (Article I)

**PEC2** Essence is difficult to learn. (Article I)

A large part of using Essence is the process of *essentializing* practices and methods and describing them with the Essence language. While this can be done in-house for project-specific practices and methods, Essence encourages the creation of collections of practices, or methods, for public use. This can be done e.g., through the Essence Practice Library (Ivar Jacobson International, n.d.). In Article II, we compile an extensive list of startup practices. This list is categorized under the Essence alphas, and can serve as a basis for describing these practices formally by using the Essence language. Moreover, we look at how the practices fit under the Essence alphas by categorizing them into the seven alphas. In the process, we propose new potential alphas for startups that future research may look into.

Our findings regarding Essence in Article II are summarized in these two PECs:

**PEC3** The Essence kernel would need additional alphas to better cover the various startup practices more focused on business elements. (Article II)

**PEC4** The startup practices listed in Article II provide ample building blocks for describing startup practices using Essence. (Article II)

Finally, in Article V, we propose a method for software startups: the Startup Cards. Due to our findings in Article I and Article III, the method is not formally described with the Essence language. Aside from what was already discussed in this section in relation to Article I, in Article III we develop a method for AI ethics: ECCOLA. ECCOLA, too, was originally described using Essence. However, during the Action Research process, through which we developed ECCOLA, we begun to think that Essence was only making the adoption of the method more complex than it needed to be, and ultimately opted to not use Essence to describe the method. As a result, using these lessons learned, the Startup Cards also do not use Essence.

Our findings related to Essence in the context of method development are summarized by the following PEC:

**PEC5** Because Essence is difficult to learn, describing a method with Essence adds an extra layer of complexity to method adoption when its users not only have to learn to use the new method, but *also* how to use Essence. (Article III and V)

Despite this being the case, much like ECCOLA, the Startup Cards are still built on the idea of essentializing practices (discussed in Section 2.5). Moreover, the Startup Cards in Article V are colour-coded into the three existing Essence areas of concern (Customer, Solution, and Endeavor), as well as a potential fourth area of concern: business (based on the results of Article II). With some effort on the parts of its users, the Startup Cards can be used as a formal Essence method.

**PEC6** The Startup Cards for Early-Stage Startups provide a card-based method that can be described with Essence, although additional work to make it Essence conformant is needed. (Article V)

### **5.1.2 Work Practices and Decision-Making in Software Startups**

Startups struggle to utilize existing SE methods, as these methods are aimed at larger organizations. Existing research argues that startups largely develop software using singular Agile practices (Paternoster et al., 2014) or even ad hoc. In Article II, we look into what these practices are. We take an existing list of practices from Dande et al. (2014), validate it with empirical data from 13 startups, and propose additional practices based on our data. These 76 practices give us a clearer picture of how startups work in practice past the more general, higher-level descriptions found in existing papers, such as that of Paternoster et al. (2014) and Giardino et al. (2016).

These practices are varied. When inserted into the context of Essence and its seven default alphas, they were split as follows: opportunity (7), stakeholders (4), requirements (11), software system (10), work (3), team (11), way-of-working

(15). Additionally, 15 were considered to not be well-suited for any existing Essence alpha. These 15 practices were business-related and financial practices, such as practices related to funding (e.g., 'fund it yourself'). To summarize the contribution of Article II, the following PEC is formulated:

**PEC7** The list of startup practices produced in Article II provides insights into how startups work in practice and can help startups pick up new work practices. (Article II)

Work and work practices and closely related to decision-making. In addition to providing further insights into practice and method use in startups, Article IV focuses on decision-making. Not much is known about the logic behind decision-making in startups in existing literature. To better understand *why* and how decisions are made in startups, particularly in relation to SE, we studied decision-making in software startups in Article IV. To find a suitable framework for explaining this phenomenon, we looked at business literature for a framework, as decision-making in business organizations in general is much more commonly studied. As a result, we utilize two business logics to investigate decision-making in Article IV.

Causal Logic and Effectual Logic are well-known ways of conceptualizing decision-making in business contexts. In Causal Logic, one has a pre-determined goal that one works towards by acquiring the needed resources or tools to achieve that goal. Causal logic is about planning and executing that plan while avoiding unexpected contingency to what extent possible. As Sarasvathy (2001) puts it, "to the extent we can predict the future, we can control it". Effectual Logic, on the other hand, is more reactive. It is about selecting between several possible goals with an existing set of resources at hand: "to the extent we can control the future, we do not need to predict it" (Sarasvathy, 2001).

In software startups, requirements elicitation, negotiation, and management in particular are mainly effectuation-driven processes. Causation is mostly used for certain activities that are more detailed and plan-based, such as requirement breakdown, estimation, analysis, and validation when the requirements are already known, at least to some extent. Similarly, when it comes to software design, causation is mostly seen in technical architectural activities in the form of optimization, with architecture-related decisions made after careful planning and with consideration in relation to trade-offs. On the other hand, much of the software design otherwise is driven by effectuation. Overall, from a business perspective, software design is driven by effectuation, while from a technical perspective it can be more causation-driven and plan-based.

This is summarized in the following PEC:

**PEC8** In startups, business-related decisions in SE are driven by effectuation, while technical SE decisions are more commonly driven by causation. (Article IV)

Testing, both system and user acceptance testing, are often causation-driven. However, effectuation-driven testing is often applicable for demonstration. Software maintenance is typically opportunistic and dealing with Technical Debt



(TD) is effectuation-driven. Technical debt occurs when short-term gain is prioritized over long-term sustainability in software development, resulting in a situation where these decisions that prioritized short-term gain in the past necessitate rework or refactoring in the future. It is common for startups to simply abandon existing systems they develop early on that end up riddled with technical debt instead of attempting to fix them. To summarize:

**PEC9** Technical debt is common in startups and startups often simply abandon existing systems plagued by high degrees of technical debt as opposed to attempting to fix them. (Article IV)

In terms of practices and methods, startups are characterized by self-defined, adaptive and opportunistic workflows. With less formalized work processes being typical earlier on, practices and processes usually evolve through the startup lifecycle. In other words:

**PEC10** Early-stage startups seldom use textbook methods and common practices. (Article IV)

However, these are generalizations and individual startups may differ in how they make decisions. To this end, startups can be categorized according to their decision-making logics. First are *effectuation-dominant* startups, which are the most common type of startups. These startups focus on internal resources and social capital. They typically focus on speed over quality and e.g., accept TD in order to move quickly. Secondly, and the second most common type of startup, is the *mixed-logic* startup. These startups operate under less uncertain conditions and may be spin-offs of established companies or startups that have already established themselves a customer base. These startups more commonly utilize traditional SE processes and practices, as their product development happens in a more predictable context. Thirdly and finally, some startups may be *causation-driven*, although such startups were not observed in our study in Article IV. These startups would focus on long-term and analytics-driven approaches in mostly using causal logic to make decisions.

The following two PECs summarize these findings:

**PEC11** Startups can be categorized into effectuation-dominant startups, mixed-logic startups, and causation-driven startups based on how they make decisions. (Article IV)

**PEC12** Most startups are effectuation-dominant. While mixed-logic startups are also common, causation-driven startups seem exceedingly uncommon. (Article IV)

### 5.1.3 Method: Startup Cards for Early-Stage Startups

The main result of this dissertation is the method in Article V: the Startup Cards for Early-Stage Startups. The method is a deck of cards, with each card describing one startup practice. The cards are based on existing literature, both academic (white) literature and practitioner (grey) literature (e.g., Ries 2011). These cards

focus on helping early-stage startups validate their ideas and solutions. They highlight the importance of making decisions based on data and changing course when that data so suggests.

There are 17 cards in total in the method. Each card is split into three parts: (1) motivation (i.e., why is this practice important), (2) what to do, and (3) common mistakes. Additionally, each card lists references as further reading. As the space on the cards is limited, the cards encourage their users to look deeper into the topic elsewhere, if and when needed. The full list of cards with brief descriptions is found in Table 2 below, while one of the cards is highlighted as an example in Figure 7. The method in its entirety is found as an appendix (Appendix I).

TABLE 2. Overview of the startup cards

#	Card Title	Description
1	Appealing Idea	Advice for idea generation.
2	Great Pitch	Advice for presenting the idea (“pitching”) briefly.
3	Validating the Appealing Idea	Advice for idea validation.
4	Get the Right Team Together	Emphasizes the importance of the startup team.
5	Create a Business Model	Advice for creating a business model.
6	Mapping the Competition	Advice for understanding the competition in the target market.
7	Establish Your Initial Way-of-Working	Jacobson et al. 2012; Paternoster et al. 2014.
8	Validating the Potential Solution	Advice for solution (product/service) validation.
9	Frequent Early Pivots	Emphasizes the importance of pivoting (changing direction) when the idea of some part of it starts looking unviable.
10	Utilize Metrics	Advice for utilizing data in the form of metrics in various ways.
11	Minimum Viable Product (in One Day)	Advice for using MVPs to validate the idea and solution.
12	Startup Spirit	Emphasizes the importance of having the mindset of an entrepreneur.
13	The Learn-Measure-Build Loop	Further advice for using MVPs in a data-driven manner.
14	Calculate the Financial Metrics	Advice on how to better convince potential investors with financial numbers.
15	Manage Scope	Advice for handling requirements and scope.
16	Work With Your (Future) Users	Emphasizes the importance of involving the user in the development process to what extent possible.
17	Make it Stable	Emphasizes the importance of basic quality even when aiming for fast time-to-market as a startup.

The cards do not form a strict process, although some activities are more relevant for earlier stage startups that are still working on figuring out the specifics of their initial idea, while others are more relevant to startups already working on a software solution to address their business idea. Each card is standalone, although the cards occasionally refer to other cards. In this fashion, the cards encourage their users to treat the cards as a checklist of issues to tackle, focusing on what they feel is the most relevant at any given time.

**Customer  
Pattern**

(2) Great Pitch

**Motivation:** Pitching means presenting your idea and your business in a concise manner. I.e., selling your idea.

**What to do:** Prepare a pitch for your startup and keep it up to date as you progress. Practice until you know it word-to-word. A pitch is usually 1 to 10 minutes. The focus of the pitch depends on your progress and your audience. Pitches typically include at least:

- (1) Problem: what problem are you solving?
- (2) Solution: how are you solving the problem?
- (3) Customer/User: who is your target audience?
- (4) Business: how do/will you make money? Numbers!
- (5) Market: how big is it? Is there competition?
- (6) Team: who are you?

Use evidence to back your claims up. Convince your audience you are not just selling them your hallucination.

**Common mistakes:** Unclear explanations. Failing to explain core aspects of your startup, e.g. business model. Not practicing your pitch beforehand.

**Sources:** [1] Colin Clark (2008) The impact of entrepreneurs' oral 'pitch' presentation skills on business angels' initial screening investment decisions, *Venture Capital*, 10(3), pp. 257-279.  
[2] Moss, J. (2018). How to Make A Winning Pitch Deck for Your Startup. Retrieved from <<https://www.forbes.com/sites/forbesynco/2018/08/28/how-to-make-a-winning-pitch-deck-for-your-startup/>>

22.10.2021

FIGURE 7. Card example from the method: startup card 2

Each card contains ideas on how to tackle the topic of the card. Sometimes, these come in the form of questions, e.g., “have you considered [thing]?” when no best practice is available. In some cases, the cards propose good or best practices. In this fashion, the cards provide ideas for how to address the topics in them. Ideally, the users of the cards would look further into the topics by using the related literature on the cards or through other resources, if needed, as the cards are A5-sized, limiting the amount of information they are able to contain.

The final result of this dissertation can be summarized as follows:

**PEC13** The Startup Cards for Early-Stage Startups method provides a way for startups to improve their way-of-working through the use of the cards. (Article V)

## 5.2 Validity Threats

In this section, the threats to validity of the articles of the dissertation are discussed. This section is structured so that each of the five articles has its own subsection for validity threats. As such, there are five subsections in this section.

### 5.2.1 Article I

In Article I, we had 102 student teams utilize Essence (Jacobson et al., 2012) while working on a practical SE project. The data used for this study came in the form of a subset of the course deliverables. Alongside other course material the student teams delivered at the end of the course was a brief survey on their use of Essence. In the report, the students were asked to discuss what was good about Essence, what was bad about Essence, and how they had utilized Essence during their project.

As the data collection relied on self-reported use, we were unable to confirm the extent of Essence’s utilization among the teams. Whether the teams really had utilized Essence to tailor a method, if they so claimed, remained unknown if they did not include such content into their report. On the other hand, in many cases it was possible to determine based on their responses whether the team had at least understood the specification or not. For example, some teams would discuss Essence as a method. Additionally, as the course deliverable was a report written by each team rather than each individual student, it is possible that the Essence section was simply left to the student(s) that most engaged with Essence or best understood it, as opposed to representing the entire team.

Moreover, as the teams were not required to use Essence in any specific manner, or at all, the utilization of Essence varied across the teams. The only common Essence-related task during the course was that the teams were asked to re-construct their current way of working approximately halfway through the course, using practices an online library (Ivar Jacobson Practice Library) to do so. After describing their current way of working, the teams were asked to modify it

as they best saw fit based on their team's experiences with the project up until that point. Indeed, the reported Essence experiences in many of the reports revolved around this common task, pointing to little Essence use past it. As such, the reported use experiences of some of the teams were more limited than those of others, and may not have included much experiences in using Essence as far as its project management aspects (alphas, alpha states) are considered.

Other potential threats to validity stem from how Essence was introduced to the course and the student teams. Essence was introduced through a lecture. As such, if the students nonetheless considered Essence difficult to learn, this was after being given an introductory lecture on the topic. In addition to the introductory lecture, the students were also introduced to the Ivar Jacobson Practice Library in a guided manner. The library contains established practices described using Essence, which can be used as building blocks when tailoring or creating methods with Essence. Individuals trying to learn to use Essence without such a guided introduction would likely have an even more difficult time, especially in the absence of accessible tutorial resources.

### **5.2.2 Article II**

In Article II, we conducted a qualitative case study on work practices in software startup. The aim was to understand how startups work, as well as to evaluate the suitability of the Essence Theory of Software Engineering in the startup context. In terms of the latter objective, the goal was to evaluate how the uncovered startup practices would fit the context of the Essence kernel and its alphas, and whether new alphas would potentially be needed to account for all the practices. Qualitative interviews were used to collect data from 12 startups.

The level of abstraction in describing practices is a potential threat to validity. In many cases, a practice could be further broken down into more atomic practices for further detail. For example, the Minimum Viable Product can be considered a practice. Yet different types of MVPs, either as categories of MVPs or even singular types of MVPs, could be considered practices as well. In this fashion, some information is generally omitted when describing practices, as one has to choose between detail and comprehensiveness. In the case of Article II, some of the practices could be further split into multiple, more atomic practices.

In Article II, the number of cases presents a potential threat to validity. A point of full saturation was not reached with 12 cases, even though it is arguably a satisfactory number of cases in case study research. For example, Eisenhardt (1989) considers five cases sufficient for novel research areas. Nonetheless, new practices would continue to emerge from the cases, and some practices that were not prevalent enough (only 1 case out of 12) were omitted from the list. It would seem that continuing to add more cases would have resulted in an even more extensive list of practices. Nonetheless, as the primary goal of the article was to evaluate Essence in the startup context, this was not done. The list of practices in Article II was considered sufficient for this purpose. Additionally, I argue that the way we used an existing list of practices as the basis of the study, and which we further validated in it, also dampens this threat to validity.

Finally, in this study, all the case startups were Finnish or Norwegian. This is a potential threat to validity, as it is possible for the local Nordic startup culture to differ from that of, e.g., the United States, resulting in different practices being commonly used.

### 5.2.3 Article III

In Article III, we propose a method for implementing AI ethics in practice, ECCOLA. ECCOLA was developed using AR, over the course of multiple iterations, and using multiple types of data. During the process, ECCOLA was tested in practice and the data from its use was utilized to iteratively improve the method during the process.

Among other validity threats, I discuss what Kock (2004) considers the three primary threats to AR: uncontrollability, contingency, and subjectivity. Subjectivity becomes an issue when the researcher is deeply involved with the client organization, and may result in bias, especially if there is potential for conflict of interest. In Article III, various types of organizations and data were involved in the AR process over multiple years. In the early iterations, data was collected from a large number of student project teams, which presented little risk of subjectivity over researcher involvement. The researchers were not closely involved with the teams as they worked and only met the teams during weekly mentor meetings. The students were encouraged to be critical of the method if needed, and the feedback was considered honest and could be used to improve the method. Later, further company data was used to keep working on ECCOLA, but company data does not yet play a large role in Article III (as opposed to its extended version, Vakkuri, Kemell et al. (2021)).

Contingency, as Kock (2004) discusses it, is largely synonymous with external validity, or generalizability. One problem Kock (2004) associates with it is that the body of data in AR is typically “broad and shallow,” referring to a situation where there is a lot of data that may not be that valuable from a research point of view. In Article III, the chosen research approach mitigates contingency as a validity threat to some extent. The involved organizations were largely student project teams, with the addition of one small real-world blockchain project team. This made it so that there was no notable abundance of data present in the organizations. Moreover, only data related to the use of the method was collected and utilized. As such, data was collected with a clear goal in mind, limiting the scope of the study.

As for uncontrollability, the research setting of choice served to give us ample control over the organizations. Being student teams, the students would have to follow the instructions of the teaching team and the researchers. As opposed to studying a business organization, this gave us a large amount of control over the teams. We instructed the teams to utilize the ECCOLA method, which they consequently did, while also producing data of its use. Uncontrollability did, thus, not present notable threats to validity in this context.

Additional threats to validity related to AI ethics could be discussed in relation to Article III, but I feel that they are outside the scope of this dissertation.

Further discussion on validity threats for Article III can be found in its extended journal version: Vakkuri, Kemell et al. (2021).

#### 5.2.4 Article IV

In Article IV, we studied decision-making in software startups by means of a multiple case study. As the theoretical framework for the study, we used causal logic and effectual logic. The two types of logics were applied while looking at decision-making related to SE decisions. As data, we used qualitative interview data from 40 startups. Despite the data being qualitative in nature, we utilized a quantitative approach to analyze it through thematic analysis. The number of the codes was used to draw some conclusions in the article.

While discussing validity threats for this article, I refer to the framework of Runeson & Höst (2009) who posit that there are four types of validity to consider: construct validity, internal validity, external validity, and reliability. This framework is not used through the entirety of this section due to its arguably positivistic nature. For example, internal validity, as Runeson & Höst (2009) consider it, suits relativistic case studies and action research poorly. On the other hand, it is a suitable framework for discussing Article IV.

In Article IV, the research approach was built on existing studies. The components used were based on existing research, and the measure of entrepreneurial logic was based on approaches reported in previous studies (Reymen et al., 2015; McKelvie et al., 2020). This was done to account for construct validity. From the point of view of internal validity, this study did not aim to determine relationships between the studied components, and as such this particular threat to validity is not of notable concern in Article IV. Nonetheless, the results of the study were compared to existing literature (e.g., Giardino et al., 2016; Hevner & Malgonde, 2019; Melegati et al., 2019), and similarities, contrasts, and explanations were examined in the light of extant research. These comparisons can be argued to have enhanced the internal validity of Article IV.

In terms of external validity, or generalization, there are some threats to validity. The number of cases in Article IV is high, 40 startups. This also made it possible to look at the results more quantitatively. On the other hand, the case startups were primarily based in Norway and the other Nordic countries. The case startups were also mostly early or mid-stage startups and did not include startups who had reached later, scale-up stages. The team sizes of the startups included in the study were also small (between 3 and 20), and the case startups were largely funded by bootstrapping. As such, these findings are most applicable to startups with similar characteristics. Nonetheless, the large sample size does provide Article IV with some generalizability. Moreover, as this was not a longitudinal study, we cannot provide much discussion on how entrepreneurial logics might change over time in the same startup as a result of various factors.

To tackle validity threats related to reliability, all case startups were invited to proofread the part of the results they contributed to, in order to ensure its conformance with reality. We, the authors, also discussed the results over several

rounds of discussion after and during the data analysis, in order to tackle any over-interpretation, and to account for alternative interpretations. The first and second author also cross-checked the results of the analysis. The review process of the journal Article IV was submitted to and accepted to, *Empirical Software Engineering*, also helped increase the reliability of the study.

### 5.2.5 Article V

Article V presents the primary contribution of this dissertation: a method for software startups. This card-based method was developed iteratively using an AR approach. Over the course of 4 years, data from 40 startups was collected in a practical course setting. Learning diaries formed the bulk of the collected data, but other types of supporting data were also used. During the AR process, the method was iteratively improved based on the data.

Among other validity threats, I discuss the three threats to AR validity discussed by Kock (2004): uncontrollability, contingency, and subjectivity. In the case of Article V, the threat of subjectivity has relatively little relevance for multiple reasons. First, the AR process involved a large number of organizations, resulting in more shallow interaction with the involved startups. In fact, and secondly, the approach was rather hands-off, with the researchers only being involved with the startups through weekly mentor meetings. As such, I retain that the collection of the data, as well as the research process itself, was sufficiently objective. It is only in the analysis phase that subjectivity a more relevant threat to validity. In the article, some quantification was added to the analysis of the otherwise qualitative data for a more transparent analysis.

Uncontrollability is a common potential threat in AR. The researcher never has complete control over the research environment, it being an existing organization. Moreover, sometimes change may happen in unexpected ways, and in some cases the researcher may be forced to abandon the research site before the study is finished. This is the primary threat in Article V, out of the three discussed by Kock (2004). Due to the AR approach having been more hands-off, the level of control over the organizations was lower as well. While this was a conscious choice in research design, it nonetheless contributed to uncontrollability in Article V. The startups in Article V were never required to use the method, although its use was regularly advocated by introducing the cards over the duration of the course, and occasionally the use of the method was discussed during the mentor meetings. As such, in some cases, the startups simply did not use the method – which was considered relevant data in and of itself.

On the other hand, the course setting gave us more control over the AR setting than we otherwise would have had. It made it simpler for us to stay in contact with the startups on a regular basis. Moreover, the power dynamic between the researchers (teachers) and the startup team members (mostly students) made the startups more compliant. On the other hand, as established, little control was ultimately exerted over the startups either way – as far as the



study of Article V is considered. As such, a degree of uncontrollability was built into the AR design in Article V by not making the method use mandatory.

In Article V, contingency as a validity threat is tackled, to some extent, by the chosen research design. First, the study is focused on the use of the method. Only data related to the method is of interest. Secondly, we laid plans for data collection so that only specific types of data were utilized. Thirdly, the organizations in question, early-stage startups, are limited in personnel and do not produce extensive amounts of documents early on that could be studied. This is arguably more of an issue when studying, e.g., large, multinational business organizations. Fourthly and finally, observation data was not utilized, and so, with the used data limited to data created by the startups through their own reporting, the data was more focused and concise.

Past these three types of generic validity threats for AR, however, the study has its own unique threats to validity as well. Article V utilized student data. However, while students are arguably less analogous to senior developers, for example, they are not as different from startup practitioners demographically. Startup practitioners are argued to be inexperienced, and similarly startups are characterized by small team sizes and flat organization structures (Paternoster et al., 2014; Giardino et al., 2014). Moreover, while most of the teams comprised of students, the setting in Article V was a learning “through” entrepreneurship (Sirelkhatim & Gangi, 2015) type course, where the students worked on startups as though they were real, and some were indeed intended to be real startups. The only difference between the real startups and the simulated ones were the motivations of the teams. Even the teams who never intended for the startup to become a real business (as a result of determining that it is unviable during the course, or from the get-go) still interacted with real customers, developed a real MVP, and in general carried out ‘real’ startup activities. As a result, I argue that the use of student data poses less limitations in this context than it perhaps otherwise would.

Finally, the data used in Article V presents some other threats to validity. As mentioned in relation to contingency, the bulk of the data relied on self-reported use through learning diaries. As such, the data we collected was varied quantity and quality. Across 43 startups, however, I argue that we nonetheless collected a satisfactory amount of data that let us evaluate the method. It should be also noted, though, that the learning diaries were produced per team rather than per student, and as such the sentiments in the learning diaries may occasionally only present the sentiments of the person writing that part of the learning diary, as opposed to the sentiments of the entire startup team.

### **5.3 Contributions**

This section summarizes the practical and theoretical contributions of this dissertation. The first subsection discusses the theoretical contributions. The second subsection discusses the practical contributions. As mentioned in the

results section, the contributions of this dissertation are discussed through the Primary Empirical Contributions (PECs) highlighted while discussing the results. These have been organized into Table 3. While discussing the theoretical and practical contributions of this dissertation, these PECs are referred to by number (e.g., “PEC1 ...”), as they provide a concise way of summarizing the key results of the articles.

TABLE 3. Primary Empirical Contributions (PECs) of the dissertation

#	Description	Article
1	Students may be taught the idea of method tailoring through Essence.	I
2	Essence is difficult to learn.	I
3	The Essence kernel would need additional alphas to better cover the various startup practices more focused on business elements.	II
4	The startup practices listed in Article II provide ample building blocks for describing startup practices using Essence.	II
5	Because Essence is difficult to learn, describing a method with Essence adds an extra layer of complexity to method adoption when its users not only have to learn to use the new method, but also how to use Essence.	III & V
6	The Startup Cards for Early-Stage Startups provide a card-based method that can be described with Essence, although additional work to make it Essence conformant is needed.	V
7	The list of startup practices produced in Article II provides insights into how startups work in practice and can help startups pick up new work practices.	II
8	In startups, business-related decisions in SE are driven by effectuation, while technical SE decisions are more commonly driven by causation.	IV
9	Technical debt is common in startups and startups often simply abandon existing systems plagued by high degrees of technical debt as opposed to attempting to fix them.	IV
10	Early-stage startups seldom use textbook methods and common practices.	IV
11	Startups can be categorized into effectuation-dominant startups, mixed-logic startups, and causation-driven startups based on how they make decisions.	IV
12	Most startups are effectuation-dominant. While mixed-logic startups are also common, causation-driven startups seem exceedingly uncommon.	IV
13	The Startup Cards for Early-Stage Startups method provides a way for startups to improve their way-of-working through the use of the cards.	V

### 5.3.1 Theoretical Contributions

Theory-wise, this dissertation furthers our understanding of how startups develop software. Software startup research is still a young area of research in SE (Unterkaalmsteiner et al., 2016), and has not gained much ground in IS literature. Articles II and IV both discuss software development in software startups. The focus in Article II was on software startup *practices* while the focus in Article IV was on decision-making in software startups, although with a focus on software-related decisions (even in terms of business decisions). Article V, on the other hand, presents a method for early-stage software startups.

In addition to software startups, this dissertation presents some theoretical contributions in the context of the Essence Theory of Software Engineering. In this regard, Articles I, II, III, and V present practical or theoretical contributions. These practical and theoretical contributions can be considered rather intertwined, depending on how one wants to utilize them.

The list of startup practices in Article II can be used to draw some theoretical contributions, alongside the practical ones discussed in PEC7. However, these mostly serve to support conceptions found in extant literature, and as such were not highlighted as PECs. The practices in Article II were categorized using the seven Essence alphas as follows, with the numbers indicating how many practices were considered to belong under each alpha: opportunity (7), stakeholders (4), requirements (11), software system (10), work (3), team (11), way-of-working (15). Additionally, 15 were considered to not be well-suited for any existing Essence alpha, as I discuss later in this section in relation to PEC3.

The frequency of some of these practices can be used to support existing conceptions related to software startups. First, the team-related practices emphasized the importance of the team. The majority of the case startups discussed small team sizes focused on competence. This is in line with extant research that considers the startup team the key resource in startups (Cooper et al., 1994; Kemell, Elonen et al., 2020; Seppänen et al., 2017; Seppänen, 2020) and many of the practices in Article II are also related to teams. Startups are typically associated with a lack of resources (Paternoster et al., 2014), which makes the team the one resource they *do* have. As the anecdotal wisdom in various startup ecosystems posits, the team is more important than the idea, as ideas are only worthwhile if a team can execute them. To this end, flat organization structures and self-organizing teams were common practice. This is in line with Agile literature that finds self-organizing teams beneficial in Agile (Karhatsul et al., 2010).

Startups are also known to prefer various Agile practices instead of textbook methods (Paternoster et al., 2014; Giardino et al., 2016). PEC10 also highlights this based on Article IV, where we argue that based on our data from that study, too, early-stage startup seem to work unsystematically and that even these Agile practices only become more common later in the startup lifecycle. Indeed, Agile, for example, is not well-suited for startups because it focuses more on *how* to develop software while startups also struggle with *what* to develop and *why* (Bosch et al., 2013). Eight of the 13 case startups in Article II also tailored common agile practices to suit the culture and needs of their startup, with the remaining five, then, seemingly not using any for the time being.

Practices related to scoping and MVP or prototype use were common in the requirements-related practices of Article II. One of the startup anti-patterns of Klotins et al. (2019) is related to scoping issues with MVPs, also highlighting the importance of these practices. Startup literature in general, including that of practitioner experts Ries (2011) and Blank (2013), discusses the importance of focusing on the core features of the product in order to test it in practice as quickly as possible.

Overall, Article II provides a detailed look into the practices utilized by software startups. Existing studies mainly look at the bigger picture when studying software development. For example, Giardino et al. (2016) propose the Greenfield Startup Model to describe, on a high level of abstraction, how software startups develop software. Paternoster et al. (2014) discuss software development in software startups from the point of view of method use, focusing on whether startups utilize methods or established practices at all, instead of focusing on *which* practices they utilize. In comparison, Article II provides a tangible list of practices utilized by software startups.

As was the case with some of the findings of Article II, some of the findings of Article IV (PEC8; PEC9; PEC10) also support various findings in existing literature, while also providing some new insights into these issues. Technical debt is commonly associated with startups (Besker et al., 2018; Bosch et al., 2018; Giardino et al., 2016). Article IV provides insights on how startups deal with technical debt in practice. It seems to be common for startups to simply discard existing systems or components riddled with technical debt instead of attempting to refactor them (PEC9). Whether this is good practice or bad practice remains an open question, however.

PEC8 provides both novel findings and validates existing research. In existing literature, little is known about how and why decisions are made in startups. According to existing research, startups are characterized, in this regard, by the strong presence of entrepreneurial personalities, behaviors, decision-making, and leadership (Bygrave et al., 1991). The small team sizes typically seen in startups, along with other factors such as uncertainty, contribute to increasing the influence key personnel such as the CEO or CTO have on the success of the startup (Berg et al., 2018; Giardino et al., 2014; Paternoster et al., 2014). The influence of entrepreneurial personalities have been discussed in extant literature in IS (e.g., in Ojala 2015; 2016), but seldom in SE and in the startup context. In the startup context, existing studies argue that the background of the entrepreneur influences how MVPs are developed in the startup (Tripathi et al., 2018), and that the founders of startups strongly influence how requirements engineering is carried out (Melegati et al., 2019).

In this light, Article IV, with PEC8 and PEC11, provides a novel theoretical contribution. These findings help us understand the logics behind decision-making in software startups, through the lens of entrepreneurial logics, causation and effectuation. However, on a general level, the prevalence of effectuation-driven decision-making in startups can be considered to also validate existing literature.

Traditional SE can largely be likened to causal logic. SE projects are seen as a linear process with a clear goal, even if work inside the project is now often carried out iteratively. Yet startups work in more tumultuous contexts categorized, among other factors, by rapidly changing business and working environments that are multiple-influenced (Giardino et al., 2014b, Giardino et al. 2016; Bajwa et al., 2017). Startups are also associated with a lack of resources and time pressure (Paternoster et al. 2014). The startup context, thus, could be

assumed to be an environment to foster effectuation-driven decision-making, which does seem to be the case (PEC8).

On the other hand, PEC8 also provides insights into how these two logics are used in startups. Effectuation is the dominant logic behind business decisions, while causation is common in software-related decisions. Perhaps causation is prevalent in these types of decisions precisely because SE projects are often seen linear processes with a clear goal – even when iterative approaches are used. Existing SE approaches would also reflect this reality, and if startups try to utilize existing practices to what extent they can, this might result in causation logics becoming more prevalent in this context.

The main theoretical contribution of Article IV is the taxonomy for categorizing startups based on decision-making logics. This provides a framework that can be used to study startups in future studies. In the context of the framework, we also argue that most startups are effectuation-dominant, and that while mixed-logic startups are also common, causation-driven startups seem exceedingly uncommon (PEC12).

In addition to these contributions related to software development in startups, this dissertation presents multiple theoretical contributions related to the Essence Theory of Software Engineering. The list of practices in Article II was categorized under the Essence kernel alphas. In the process, we found 15 practices that were poorly suited for the existing alphas, resulting in PEC3.

PEC3 highlights the way business is closely intertwined with SE in software startups, as with a single product, the entire business of a startup hinges on that one piece of software. To this end, Klotins et al. (2019) also suggest that many business-related issues in software startups may in fact stem from SE issues. In traditional SE, software is usually developed in *projects*, although recently *continuous* SE, SaaS, and DevOps blur the line between development and operations and maintenance. It is now exceedingly rare for a software to be ‘finished’, as it is continuously developed further during its operational life. Nonetheless, with startups being largely focused on developing one product, business becomes directly linked with SE, as the viability of the company depends on that one development endeavor. Thus, that one service *is* the business. Similarly, early on, due to the flat organizational structure and small team sizes commonly seen in startups (Paternoster et al., 2014), the developers may also be closely associated with the individuals in charge of business elements or may even be working on them as well (e.g., a programming-oriented startup founder).

The Essence Theory of Software Engineering posits that its kernel includes all the elements present in every single SE endeavor. However, for the startup context, incorporating some business aspects may be necessary to achieve this. While the alphas in the customer area of concern of the kernel (stakeholders, requirements) do account for some business-oriented practices related to, e.g., validation, startups also engage in business model development. Finances and funding are also important for startups, and are, in many cases, directly related to the one software being developed (although it is also common for startups to

engage in commissioned projects unrelated to their actual, planned business to provide themselves with an income early on etc.).

Based on Article II (PEC3), we propose a fourth, business-related area of concern for Essence that would account for practices related to business model development. These alphas could be, for example, funding, business model, and marketing. However, Essence posits that alphas should be as orthogonal as possible to avoid overlap (Ng, 2015). This poses challenges for devising new alphas. Business or finance-related alphas might overlap with the ones in the customer area of concern. When the software being developed *is* the business, having the business model as an alpha has some overlap with requirements, for example, and marketing activities are certainly also relevant from the point of view of the existing stakeholder alpha. However, as Article II did not formally develop these alphas, these are theoretical contributions that further research needs to build on. In this regard, of course, it is certainly possible to simply use business-oriented tools for the more business-related issues instead and to consider them entirely out of scope of Essence even in the startup context, although this, then, diminishes the value of Essence in that context.

Articles I, III, and V together present some theoretical and practical contributions for Essence as well. Essence is difficult to learn (PEC2), and because it is difficult to learn, describing a method with Essence adds an extra layer of complexity to method adoption when its users not only have to learn to use the method, but also Essence (PEC5). Arguably, this is mostly a practical contribution of interest to those working on Essence or working with Essence. However, those interested in utilizing Essence in research should keep these potential issues in mind and think of ways to address them. This is also a theoretical contribution in the sense that ways to use Essence in a more lightweight fashion could be developed, although as Essence is an OMG standard (Object Management Group, 2018), notable changes to the specification are unlikely.

### 5.3.2 Practical Contributions

The primary practical contribution of this dissertation is the Startup Cards method presented in Article V (PEC13). Using AR, the method has been developed in a practical student project setting over the course of multiple iterations. The method is discussed in more detail in Section 5.1.3 and Article V and can be found in the appendix in its entirety. The method can help early-stage software startups make better use of their resources by utilizing the established practices described in the cards. The cards focus on highlighting the importance of validating a business idea and the related software solution, as opposed to working based on assumptions. As startups struggle to utilize existing SE methods, this method may help startups work in a more systematic fashion.

As an additional contribution, the method also provides a framework for teaching software startup entrepreneurship. It has been developed in a practical startup entrepreneurship course and can arguably be used as a tool in other such courses. The cards are not intended form a linear process in practical use (as, e.g., pivots can result in the need to repeat various activities), but the order in which

the practices are introduced can be used as a framework for teaching purposes. The order of the cards is based on startup life cycle models (e.g., Wang et al., 2016).

Finally, the version of the cards presented in Article V is the first version of the cards to be published. In this version presented here, the focus of the method has been on earlier stages of the startup life cycle, and on validation activities specifically. The method, in this state, is intended to encourage startups to validate their business idea before, and during, development, in order to help determine whether the product or service has market potential, and in what shape or form. The current card contents reflect this focus, and in terms of validation activities, the method is considered complete based on our current data and current understanding of software startup research.

On the other hand, while the method contains some SE practices, the current set of cards is more focused on validation or requirements-related issues as opposed to technical software issues. In this regard, moving further along the startup life cycle, additional cards related to more technical development activities could be proposed and included. Such cards are not included in this initial version of the deck due to 1) potential scoping issues, and 2) the study design of Article V. We studied early-stage startups in Article V, and few of the startups proceeded with development past simple mock-up MVPs during the study. While this provided a suitable setting for studying the validation-focused card deck, it was not a suitable setting for studying software development issues in startups. Additional cards related to more technical SE issues would be a suitable addition for the method, as long as the scope of the method does not become too large as a result. This is a potential future research direction we are exploring.

In addition to the method, the list of practices compiled in Article II (PEC7) can be useful for startups. The list contains 76 practices for startups. In addition to serving as a starting point for describing startup practices using Essence (PEC4), the methods can also provide startups with ideas on new practices to pick up to support their work.

Originally, we planned on describing the methods in Articles III and V with the Essence language. However, following the lessons learned in Article III, and during the AR process in Article V itself (PEC5), the cards are no longer described using Essence, although they retain some Essence elements. These include the card-based nature of the method and the colour-coding on the cards, as well as the idea behind essentializing practices in this fashion. Because of this, it is possible to use the method via Essence, although this would require notable extra effort on the part of its users (PEC6). The method assumes the existence of a fourth, business-related area of concern, and has no alphas and alpha states of its own.

To this end, PEC1 and PEC5 highlight Essence is largely a tool for established and larger software organizations. As far as I am aware, many of the companies utilizing Essence out on the field are indeed large multinational organizations. Essence is a complex specification, and has its uses in method

engineering, but it seems to be best suited for experts of that area. Introducing Essence to individuals or developers unfamiliar with modeling languages will result in a steep learning curve, and whether this is worth it remains an open question. In Article III and V, the Essence elements were simply confusing to the users of the methods, even though the some of the users in Article III had a rudimentary understanding of Essence. Based on this, I stress that *describing methods with Essence seems to only be useful for organizations already using Essence*. One cannot learn Essence by using an Essence method. Essence needs to be studied.

As for studying Essence, PEC1 presents some practical contributions in the context of Essence. Students can be taught the idea of method tailoring through Essence. However, this is not something unique to Essence, and the idea of method tailoring can certainly be taught by other means as well. In fact, considering that Essence is difficult to learn (PEC1; PEC5), one must weigh whether it is worth the effort to use Essence for this purpose. In the study of Article I, we also saw some of the student teams remark that they could see the value of Essence but felt that they could not fully utilize it due to their inexperience and lack of knowledge on SE methods. Many of the students only knew SCRUM, and teaching them to use Essence mostly resulted in them devising various ScrumButs. These lessons learned, however, may have been beneficial to them in the future.

To potentially address the difficult adoption of Essence, better tutorial resources are needed. Moreover, Essence is described poorly in many of the existing materials. The specification uses its own jargon which is difficult to understand for potential new users. More concise and understandable introductions to the specification are needed to facilitate its adoption, as we discuss in Article I as well. In Article I, the student teams struggled to find tutorial resources, and lamented the fact that seemingly the only way to really start utilizing Essence was to read a 300-page-book full of Essence jargon. For example, Essence still does not have a Wikipedia entry as of February 2022.

In terms of Essence, Article I also presents some practical implications. Most importantly, Article I highlights the importance of good tutorial resources. Students struggle to utilize Essence when the only way to familiarize themselves with the notation and the kernel is to read a 300+ pages long book. Shorter, more concise resources for beginners help newer users get into the topic. While more Essence content has been published since the writing of Article I, there is still, e.g., no Wikipedia page for Essence. This can make the method seem difficult to approach. On the other hand, past its potentially difficult adoption process, Essence can be useful for teaching students about methods. Essence is built around the idea of tailoring methods to suit the present context, which it can used to teach. While some proponents of Scrum, for example, may argue that creating ScrumButs is bad practice, tailoring methods to better suit the context at hand, when done with careful consideration as opposed to simply omitting practices to do less work, should not be treated as such.



Finally, it can be briefly noted that Article III presents a novel and valuable contribution for AI ethics by proposing a method for implementing AI ethics in practice. Thus far, AI ethics research has been characterized by a lack of empirical studies. While this is valuable work, and this existing body of knowledge was used to construct the ECCOLA method presented in Article III, there has been a prominent gap between research and practice in the area. Article III is expanded on in Vakkuri, Kemell, et al. (2021). However, this is out of the scope of this dissertation, as the role of Article III in this dissertation were its lessons learned in method engineering that supported the development of the early-stage startup method presented in Article V.

### 5.3.3 Limitations

The primary limitation of this dissertation as a whole is that all the Articles included (I-V) utilized qualitative research approaches. The generalizability of qualitative research is always a potential limitation. Whereas quantitative results are based on larger sets of data, qualitative research often generalizes, e.g., based on a handful of case companies. However, in this dissertation, this limitation has been mitigated to some extent with sizeable data sets – at least in the context of qualitative research.

In Article I, we had data from over 100 student teams. In Article IV, we had interview data from 40 case startups. In Article V, we developed the Startup Cards method over the course of four years, using data from 43 startup teams. The number of teams or startups in these three articles lends support to the generalizability of the results. For comparison, Eisenhardt (1989) argues that, for novel research areas, five cases, or even a single in-depth case, may be sufficient. Software startups as a research area in SE is no longer particularly novel, but neither is it particularly established (Unterkalmsteiner et al., 2016).

In Article II, we utilized data from 13 cases, which, compared to the aforementioned articles, is far less. Indeed, the number of cases is a particular limitation for Article II. A larger number of cases may see new practices emerge using the same research approach, and the results of Article II are not at all exhaustive.

Another limitation with Articles II and V is the lack of an established definition for what is a startup. In the lack of such widely accepted definition, the studies included any company that considered itself a startup. Thus, if imposing a specific definition for what is a startup on the data sets of these two studies, one might exclude some of the startups now included. In Article IV, we utilized a specific set of criteria for case inclusion or exclusion. While one may disagree with the criteria, they draw a clear line for what was or was not considered a startup in Article IV. While this arguably is not a large limitation to the generalizability of the results, it is useful to acknowledge that it can be difficult to differentiate between startups and other business organizations in practice.

### 5.3.4 Future Research Suggestions

Articles II and V offer future research avenues for Essence in relation to software startups. Article II provides initial evaluation of Essence in a startup context and provides preliminary suggestions for new potential alphas. Future studies could investigate whether these additional alphas really are needed, and if so, such studies could propose such formal alphas. In Article II, we provide some starting points for such a study. Article V, on the other hand, provides a method that was originally described using Essence. However, during its development process, the role of Essence was lessened. A future study could look at the method through Essence to make the method more useful to those capable of utilizing Essence. Article II provides a list of startup practices that can be expanded upon, categorized, and otherwise further studied outside the context of Essence as well.

Article IV was a cross-sectional view into startups. There we no longitudinal aspects to the study. As such, future studies could expand upon the findings of Article IV by taking on a longitudinal approach. This would make it possible to look into how and why shifts in decision-making logic happen as a result of different changes to the startup context at hand. Additionally, future studies could look into developing methods that take into account the largely effectuation-driven approach to SE that software startups seem to exhibit.

## YHTEENVETO (SUMMARY IN FINNISH)

Tässä väitöskirjassa tutkin ohjelmistokehitystä startup-yrityksissä. Väitöskirjani tavoite oli paitsi auttaa meitä ymmärtämään paremmin miten startup-yrityksen kehittävät ohjelmistoja, myös kehittää startup-yrityksille suunnattu ohjelmistokehitysmenetelmä. Tämä menetelmä on tämän väitöskirjan pääasiallinen tulos ja se on esitelty sen viidennessä artikkelissa.

Startup-yritykset eroavat monin tavoin muista yrityksistä myös ohjelmistokehityksen suhteen. Startup-yritysten on muun muassa havaittu suosivan erilaisia lähestymistapoja ohjelmistokehitykseen. Ne eivät esimerkiksi juuri käytä muiden yritysten suosimia ohjelmistokehitysmenetelmiä vaan käyttävät korkeintaan yksittäisiä alalla hyväksi todettuja työkäytänteitä (good practice) ja työskentelevät pitkälti niin kuin itse parhaaksi näkevät.

Valtaosa startupeista kuitenkin epäonnistuu. Nämä epäonnistumiset johtuvat useista eri tekijöistä. Olemassa olevassa tutkimuksessa on kuitenkin argumentoitu, että moni epäonnistuminen johtuu muun muassa ohjelmistokehitykseen liittyvistä tekijöistä. Startup-yrityksillä on etenkin ongelmia vaatimusmäärittelyn kanssa. Monet startup-yritykset kehittävät ohjelmistoa, jolla ei välttämättä ole lopulta mitään markkina-arvoa. Tällaiset ongelmat johtuvat usein siitä, että startup-yritys ei ole tehnyt riittävää markkinatutkimusta tai ollut riittävästi yhteydessä potentiaaliseen käyttäjäkuntaansa.

Tässä väitöskirjassa kehitetyn menetelmän tavoite on auttaa startup-yrityksiä välttämään muussa tutkimuksessa havaittuja haasteita ja huonoja käytänteitä. Menetelmän on myös tarkoitus painottaa idean ja tuotteen validoinnin (validation) merkitystä. Menetelmä muun muassa rohkaisee käyttäjiään aktiivisesti testaamaan oletuksiaan tuotteestaan ja ideastaan keräämällä dataa. Sen sijaan, että startup-yritys kehittäisi tuotettaan keskenään, olisi tärkeää, että potentiaalista käyttäjäkuntaa kuultaisiin jo aikaisessa vaiheessa. Toimimalla näin voisi olla mahdollista huomata jo aikaisessa vaiheessa, että tuotteella ei välttämättä olekaan kysyntää sen suunniteltujen käyttäjien keskuudessa.

Alun perin tämän väitöskirjan tarkoitus oli tutkia Essence-teorian (The Essence Theory of Software Engineering) soveltuvuutta ohjelmistoalan startup-yritysten kontekstiin ja muuttaa sitä paremmin siihen soveltuvaksi siltä osin kuin olisi tarve. Lisäksi tavoitteena oli kehittää ohjelmistoalan startupeille suunnattu ohjelmistokehitysmenetelmä käyttäen Essenceä sen mallintamisessa. Essence on käytännössä mallinnuskieli, jolla mallinnetaan työtapoja ja -menetelmiä. Mallinnuskielen lisäksi Essenceen kuuluu kuitenkin ns. kerneli (kernel), joka käytännössä sisältää erilaisia rakennuspalikoita, joiden päälle ja joita käyttäen menetelmiä Essencellä mallinnetaan.

Tutkimusprosessin aikana kuitenkin alkoi vaikuttaa siltä, että Essence ei ollut sopiva työkalu tähän käyttötarkoitukseen. Artikkelissa I huomasimme jo, että Essenceä oli vaikea oppia käyttämään. Kun artikkelissa kolme käytimme Essenceä menetelmän kuvaamiseksi, huomasimme nopeasti, että menetelmää vaikea oppia käyttämään. Käyttääkseen menetelmäämme, oli sen käyttäjien ensin opittava käyttämään ja ymmärtämään Essenceä. Tämä teki menetelmän oppimisesta

huomattavan työstä sen käyttäjille. Kehittäessämme tätä kolmannessa artikkelissa esiteltyä menetelmää iteratiivisesti, vähensimme Essencen roolia menetelmän kuvaamisessa iteraatioiden välillä, kunnes lopulta sen rooli oli hyvin pieni. Näiden tulosten perusteella päätimme myös olla käyttämättä Essenceä artikkelissa viisi esitellyn menetelmän kuvaamiseksi. Essence jäi näin ollen väitöskirjasani lopulta pienempään rooliin kuin alun perin oli suunniteltu. Menetelmässä on kuitenkin edelleen vaikutteita Essencestä.

Seuraavaksi käsittelen väitöskirjan tuloksia artikkelitasolla. *Artikkelissa I* tutkimme Essencen soveltuvuutta pienempiin ohjelmistokehitysorganisaatioihin. Essence on lähinnä suurten ohjelmistokehitysyriyten käytössä, vaikkei sitä ylipäätään usein käytetä ainakaan toistaiseksi. Artikkelin yksi tutkimuksessa tutkimme opiskelijatiimien käyttökokemuksia Essencestä. Tutkimuksessa 102 opiskelijatiimiä käyttivät Essenceä osana käytännönläheistä ohjelmistokehitysprojektiä, jossa tiimit kehittivät ohjelmistoa yliopistokurssia varten. Tutkimuksen perusteella myös opiskelijatiimien onnistui käyttää Essenceä, mutta heidän oli kuitenkin vaikea oppia sitä käyttämään.

*Artikkelissa II* tutkii myös Essenceä. Tässä artikkelissa Essenceä tutkittiin startup-yriyten kontekstissa. Käyttämällä pohjana muiden tutkijoiden luomaa listaa startup-yriyksissä käytetyistä työkäytänteistä, loimme listan 76 startup-yriyten käyttämästä työkäytänteestä haastatteleamalla startup-yriyksiä. Listan perusteella voimme ymmärtää paremmin, miten startup-yriyksissä kehitetään ohjelmistoja. Lisäksi listaa katsottiin Essencen näkökulmasta. Tutkimuksen perusteella ehdotimme muutoksia Essenceen, jotka tekisivät siitä paremmin startup-kontekstiin soveltuvan.

*Artikkelissa III* esittelemme menetelmän (ECCOLA) tekoälyn etiikan tuomiseksi käytäntöön. ECCOLAn kehityksestä opimme useita asioita, jotka auttoivat viidennessä artikkelissa esitellyn startup-menetelmän kehittämisessä. Vastaavasti startup-menetelmän kehitys tuki ECCOLAn kehitystä samalla tavalla, sillä kumpaakin menetelmää kehitettiin samaan aikaan vuosina 2018–2021. Molemmat menetelmät (1) käyttävät kortteja menetelmän esitystapana, (2) kehitettiin syklisiä toimintatutkimusta (Cyclical Action Research) tutkimusmenetelmänä käyttäen, (3) oli alun perin määrä kuvata käyttämällä Essenceä ja (4) ovat modulaarisia ja tukevat iteratiivista ohjelmistokehitystä. Näin ollen menetelmien kehitysprosessit tukivat toisiaan.

*Artikkeli IV* sen sijaan tutkii päätöksentekoa ohjelmistoalan startup-yriyksissä. Tutkimuksessa käytettiin teoreettisena viitekehystenä liiketoimintatutkimuksen alalla käytettyä viitekehystä, jossa yriyten päätöksenteko jaetaan kehittämisenlogiikkaan (effectual logic) ja suunnittelulogiikkaan (causal logic). Käytännöllisenä esimerkkinä näiden havainnollistamiseksi käytettäkään kokkia, joka tekee ruokaa. Suunnittelulogiikkaan nojaava kokki katsoisi reseptistä tarvittavat ainekset ja kävisi sitten kaupassa ostamassa ne. Kehittämisenlogiikkaan nojaava kokki sen sijaan katsoisi ensin jääkaappiin ja päättäisi sitten tilannepohjaisesti, mitä aikoo valmistaa. Vastaavaa tutkimusta ei ole juuri tehty startup-yriyten kontekstissa, eikä etenäkään ohjelmistokehitykseen liittyen startup-kontekstissa.

Artikkelissa IV tutkimme startup-yritysten päätöksentekoa ohjelmistokehitykseen liittyvissä päätöksissä tämän viitekehysten näkökulmasta.

Tutkimuksen perusteella muodostimme typologian, joka jakaa startup-yritykset kolmeen ryhmään sen mukaan, miten ne tekevät päätöksiä näiden kahden päätöksentekologiikan näkökulmasta. Tämän typologian mukaan startupit voivat olla (1) kehittämispainotteisia (effectuation dominant), (2) monilogiikkaisia (mixed logic) tai (3) suunnittelupainotteisia (causation-dominant). Suurin osa startupeista vaikuttaisi olevan kehittämispainotteisia. Monilogiikkaiset startupit ovat myös yleisiä. Sen sijaan suunnittelupainotteisia startuppeja ei tutkimuksen 40:n startupin joukosta löytynyt. Lisäksi artikkelissa esiteltiin useita löydöksiä liittyen startup-yritysten ohjelmistonkehitys-käytäntöihin.

*Artikkeli V* esitteli tämän väitöskirjan tärkeimmän tuloksen: startup-yrityksille suunnatun ohjelmistonkehitysmenetelmän. Tämä menetelmä on korttipohjainen menetelmä, jossa jokainen kortti esittelee yhden tärkeän aiheen tai suoranaisen työkäytännön. Korttipakassa on yhteensä 17 korttia. Kortit perustuvat paitsi tämän väitöskirjan muihin artikkeleihin myös väittelijän muihin tutkimuksiin (joita esiteltiin luvussa 1.4) sekä tieteenalan muuhun tutkimukseen. Kortit perustuvat myös muuhun startup-aiheiseen, asiantuntijoiden kirjoittamaan kirjallisuuteen akateemisen kirjallisuuden lisäksi.

## REFERENCES

- Abrahamsson, P., Suoranta, M., Lahti, S., & Kemell, K.-K. (2021). The Startup Scratch Book : Opening the Black Box of Startup Education. In E. Klotins, & K. Wnuk (Eds.), *ICSOB 2020 : 11th International Conference of Software Business* (pp. 193-200). Springer. Lecture Notes in Business Information Processing, 407. [https://doi.org/10.1007/978-3-030-67292-8\\_15](https://doi.org/10.1007/978-3-030-67292-8_15)
- Alavi, M. & Carlson, P. (1992). A Review of MIS Research and Disciplinary Development. *Journal of Management Information Systems*, 8, 45-62.
- Alshenqeeti, H. (2014). Interviewing as a Data Collection Method: A Critical Review. *English Linguistics Research*, 3(1), 39-45.
- Apa, C., Jeronimo, H., Nascimento, L., Vallespir, D., & Travassos, G. (2018). The Perception and Management of Technical Debt in Software Startups. In: Nguyen-Duc A., Münch J., Prikladnicki R., Wang X., Abrahamsson P. (eds) *Fundamentals of Software Startups*. Springer, Cham, 61-78.
- Ayala, C., Nguyen-Duc, A., Franch, X., Höst, M., Conradi, R., Cruzes, D., Babar, M. A. (2018). System requirements-OSS components: matching and mismatch resolution practices - an empirical study. *Empirical Software Engineering*, 23(6), pp. 3073-3128. <https://doi.org/10.1007/s10664-017-9594-1>
- Bajwa, S. S., Wang, X., Nguyen-Duc, A., & Abrahamsson, P. (2016). How Do Software Startups Pivot? Empirical Results from a Multiple Case Study. In: Maglyas A., Lamprecht AL. (eds) *Software Business*. ICSOB 2016. Lecture Notes in Business Information Processing, vol 240. Springer, Cham)
- Bajwa, S., Wang, X., Nguyen-Duc, A., Abrahamsson, P. (2017). "Failures" to be celebrated: an analysis of major pivots of software startups. *Empirical Software Engineering*, 22(5), 2373-2408.
- Baldridge, R., & Curry, B. (2021). What is a Startup? Forbes. <https://www.forbes.com/advisor/investing/what-is-a-startup/#544a2a9a4c63>
- Berg, V., Birkeland, J., Nguyen-Duc, A., Pappas, I., Jaccheri, L. (2018). Software startup engineering: A systematic mapping study. *Journal of Systems and Software*, 144, 255-274.
- Besker, T., Martini, A., Edirisooriya, L., Blincoe, K., & Bosch, J. (2018). Embracing Technical Debt, from a Startup Company Perspective. In Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 415-425.
- Biyani, G. (2013). Explained: The actual difference between growth hacking and marketing. <http://thenextweb.com/insider/2013/05/05/the-actual-difference-between-growth-hacking-and-marketing-explained/>
- Blank, S. (2013a). Four Steps to Epiphany. K&S Ranch.
- Blank, S. (2013b). Why the Lean Startup-Up Changes Everything. Harvard Business Review (May 2013).

- Bohnsack, R., & Liesner, M. (2019). What the hack? A growth hacking taxonomy and practical applications for firms. *Business Horizons*, 62(6), PP. 799-818.
- Bosch, J., Olsson, H., Björk, J., & Ljungblad, J. (2013). The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups. *LESS 2013. LNBIP*. 167. 1-15.
- Bourque P., & Fairley, R. E. (2014). Guide to the software engineering body of knowledge (SWEBOK (R)): Version 30. IEEE Computer Society Press.
- Braun, V., & Clarke, V. (2006) Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77-101.  
<https://doi.org/10.1191/1478088706qp063oa>
- Bubenko jr, J., (1986) Information System Methodologies - A Research View. In: Information System Design Methodologies: Improving the Practise. Proceeding of the IFIP WG 8.1 Working Conference, Noordwijkerhout, the Netherlands, 5-7 may, 1986. (eds. T.W. Olle, H.G. Sol, A.A. Verrinj-Stuart) North Holland Publishing Company, Amsterdam, pp. 289-318.
- Carmel, E. (1994). Time-to-completion in software package startups. In Proceedings of the 27th Hawaii International Conference on System Sciences (HICSS), pp. 498-507. IEEE.
- Clark, C. (2008) The impact of entrepreneurs' oral 'pitch' presentation skills on business angels' initial screening investment decisions, *Venture Capital*, 10(3), pp. 257-279.
- Conboy, K. (2009). Agility From First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research*, 20. 10.1287/isre.1090.0236.
- Cooper, A. C., Gimeno-Gascon, F. J., & Woo, C. Y. (1994). Initial Human and Financial Capital as Predictors of New Venture Performance. *Journal of Business Venturing*, 9(5), pp. 371-395.
- Crowne, M. (2002). Why software product startups fail and what to do about it. Evolution of software product development in startup companies. In Proceedings of the 2002 Engineering Management Conference IEMC'02, pp. 338-343, IEEE.
- Cruzes, D. S., & Dybå, T. (2011). Recommended steps for thematic synthesis in software engineering. In Proceedings of the 2011 Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 275-284, IEEE.
- Dande, A., Eloranta, V. P., Kovalainen, A. J., Lehtonen, T., Leppänen, M., Salmimaa, T., ... Koskimies, K. (2014). Software startup patterns - an empirical study. Tampereen teknillinen yliopisto. Tietotekniikan laitos. Raportti-Tampere University of Technology. Department of Pervasive Computing. Report; 4.
- Davison, R., Martinsons, M.G., & Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, 14(1), pp. 65-86.  
<http://dx.doi.org/10.1111/j.1365-2575.2004.00162.x>
- Digital.ai (2020). 14th annual State of Agile report. Retrieved 28 Sep 2021 from <<https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494>>.

- Donckels, R., & Segers, J. P. (1990). New Technology Based Firms and the Creation of Regional Growth Potential: Theoretical Considerations and Empirical Evidence for Belgium. *Small Business Economics*, 2(1), pp. 33-44.
- Essence in Practice (n.d.). What is Essence?  
<https://essence.ivarjacobson.com/services/what-essence>
- Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of Management Review*, 14(4), pp. 532-550.
- Ellis, S., & Brown, M. (2014). *Startup Growth Engines: Case Studies of How Today's Most Successful Startups Unlock Extraordinary Growth*. E-Book.
- Fagerholm, F., Guinea, A., Mäenpää, H., & Münch, J. (2017). The RIGHT model for Continuous Experimentation. *Journal of Systems and Software*, 123, 292-305. 10.1016/j.jss.2016.03.034.
- Feiz et al. (2021). Typology of Growth Hacking Strategies Along the Growth Hacking Funnel. *Iranian Journal of Management Studies*, 14(2), pp. 331-346.
- Fitzgerald, B. (1996) Formalized systems development methodologies: a critical perspective. *Information Systems Journal*, 6, pp. 3-23.
- Fudickar, R., & Hottenrott, H. (2019). Public research and the innovation performance of new technology based firms. *The Journal of Technology Transfer*, 44, pp. 326-358.
- Ghanbari, H. (2017). Investigating the causal mechanisms underlying the customization of software development methods. *Jyväskylä studies in computing*, 258, 2017. [Doctoral dissertation]
- Ghanbari, H., Vartiainen, T., Siponen, M. (2018). Omission of Quality Software Development Practices: A Systematic Literature Review. *ACM Computing Surveys*, 51(2).
- Ghezzi, A. (2018). Digital startups and the adoption and implementation of Lean Startup Approaches: Effectuation, Bricolage and Opportunity Creation in Practice. *Technological Forecasting and Social Change*, 146, pp. 945-960.
- Giardino, C., Bajwa, S. S., Wang, X., & Abrahamsson, P. (2015, May). Key challenges in early-stage software startups. In *International conference on agile software development* (pp. 52-63). Springer, Cham.
- Giardino, C., Paternoster, M., Unterkalmsteiner, M., Gorschek, T., and Abrahamsson, P. (2016). Software Development in Startup Companies: The Greenfield Startup Model. *IEEE Transactions on Software Engineering*, 42(6), 585-604.
- Giardino, C., Unterkalmsteiner, M., Paternoster, N., Gorschek, T., and Abrahamsson, P. (2014b). What Do We Know about Software Development in Startups? *IEEE Software*, 31(5), pp. 28-32. doi: 10.1109/MS.2014.129.
- Giardino, C., Wang, X., & Abrahamsson, P. (2014c). Why early-stage software startups fail: a behavioral framework. In *International conference of software business* (pp. 27-41). Springer, Cham.



- GoodFirms Research (2019). Remarkably Useful Stats and Trends on Software Development. <https://www.goodfirms.co/resources/software-development-research>
- Grant, D., Ngwenyama, O., & Klein, H., (1992) Validating ISD Methodologies Within The Organizational Context: An Action Research Case Study. Working paper series, Binghamton, State University of New York, 92-215.
- Graziotin, D., & Abrahamsson, P. (2013). A Web-based modeling tool for the SEMAT Essence theory of Software Engineering. *Journal of Open Research Software*, 1.
- Gutbrod, M., & Münch, J. (2018). Teaching Lean Startup Principles: An Empirical Study on Assumption Prioritization. In *Software-intensive business: start-ups, ecosystems and platforms: proceedings of the International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms (SiBW 2018): Espoo, Finland, December 3, 2018.*-(CEUR workshop proceedings; 2305) (pp. 245-253). RWTH Aachen.
- Herttua, T., Jakob, E., Nave, S., Gupta, R., & Zylka, M. P. (2016). Growth Hacking: Exploring the Meaning of an Internet-Born Digital Marketing Buzzword. In: Zylka M., Fuehres H., Fronzetti Colladon A., Gloor P. (eds) *Designing Networks for Innovation and Improvisation*. Springer Proceedings in Complexity. Springer, Cham.
- Hevner, A., & Malgonde, O. (2019). Effectual application development on digital platforms. *Electronic Marketing*, 29(3), 407-421.
- Ivar Jacobson International (n.d.). Essence Practice Library. <https://practicelibrary.ivarjacobson.com/start>
- Jacobson, I., Ng, P., McMahon, P. E., Spence, I., and Lidman, S. (2012). The Essence of Software Engineering: The SEMAT Kernel. *ACMQueue*, 10, pp. 40-52.
- Jacobson, I., & Stimson, R. (2018). Escaping Method Prison. [http://semat.org/news/-/asset\\_publisher/eaHEtyeuE9wP/content/escaping-method-prison](http://semat.org/news/-/asset_publisher/eaHEtyeuE9wP/content/escaping-method-prison)
- Järvinen, P. (2001) *On research methods*. Juvenes-Print, Tampere.
- Järvinen, P. (2004). Research Questions Guiding Selection of an Appropriate Research Method. Hansen, Bichler and Mahrer (eds.), *Proceedings of European Conference on Information Systems 2000*, 3-5 July. Vienna: Vienna University of Economics and Business Administration, 2000. pp. 124-131
- Karhatsu, H., Ikonen, M., Kettunen, P., Fagerholm, F., & Abrahamsson, P. (2010). Building Blocks for Self-Organizing Software Development Teams a Framework Model and Empirical Pilot Study. In *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE)*.
- Kemell, K.-K., Elonen, A., Suoranta, M., Nguyen-Duc, A., Garbajosa, J., Chanin, R., Melegati, J., Rafiq, U., Aldaej, A., Assyne, N., Sales, A., Hyrynsalmi, S., Risku, J., Edison, H., & Abrahamsson, P. (2020). Business Model Canvas

- Should Pay More Attention to the Software Startup Team. In A. Martini, M. Wimmer, & A. Skavhaug (Eds.), SEAA 2020 : 46th Euromicro Conference on Software Engineering and Advanced Applications (pp. 342-345). IEEE. Euromicro Conference on Software Engineering and Advanced Applications. <https://doi.org/10.1109/seaa51224.2020.00063>
- Kemell, K.K., Evensen, A., Wang, X., Risku, J., & Abrahamsson, P. (2019). A Tool-based Approach for Essentializing Software Engineering Practices. In Proceedings of the 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA2019), Kallithea-Chalkidiki, Greece, 2019, pp. 51-55.
- Kemell, K.-K., Nguyen-Duc, A., Wang, X., Risku, J., & Abrahamsson, P. (2018). The Essence Theory of Software Engineering : Large-Scale Classroom Experiences from 450+ Software Engineering BSc Students. In M. Kuhrmann, K. Schneider, D. Pfahl, S. Amasaki, M. Ciolkowski, R. Hebig, P. Tell, J. Klünder, & S. Küpper (Eds.), PROFES 2018 : Product-Focused Software Process Improvement : 19th International Conference, Proceedings (pp. 123-138). Springer. Lecture Notes in Computer Science, 11271. [https://doi.org/10.1007/978-3-030-03673-7\\_9](https://doi.org/10.1007/978-3-030-03673-7_9)
- Kemell, K.-K., Ravaska, V., Nguyen-Duc, A., & Abrahamsson, P. (2020). Software Startup Practices : Software Development in Startups Through the Lens of the Essence Theory of Software Engineering. In M. Morisio, M. Torchiano, & A. Jedlitschka (Eds.), PROFES 2020 : 21st International Conference on Product-Focused Software Process Improvement, Proceedings (pp. 402-418). Springer. Lecture Notes in Computer Science, 12562. [https://doi.org/10.1007/978-3-030-64148-1\\_25](https://doi.org/10.1007/978-3-030-64148-1_25).
- Kemell, K.-K., Risku, J., Strandjord, K. E., Nguyen-Duc, A., Wang, X., & Abrahamsson, P. (2020). Internal Software Startups : A Multiple Case Study on Practices, Methods, and Success Factors. In A. Martini, M. Wimmer, & A. Skavhaug (Eds.), SEAA 2020 : 46th Euromicro Conference on Software Engineering and Advanced Applications (pp. 326-333). IEEE. Euromicro Conference on Software Engineering and Advanced Applications. <https://doi.org/10.1109/seaa51224.2020.00061>
- Kemell, K.-K., Wang, X., Nguyen-Duc, A., Grendus, J., Tuunanen, T., & Abrahamsson, P. (2018). 100+ Metrics for Software Startups : A Multi-Vocal Literature Review. In S. Hyrynsalmi, A. Suominen, C. Jud, X. Wang, J. Bosch, & J. Münch (Eds.), SiBW 2018 : Proceedings of the First International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms (pp. 15-29). CEUR workshop proceedings, 2305. RWTH Aachen University. Retrieved from <http://ceur-ws.org/Vol-2305/paper02.pdf>
- Khurum, M., Fricker, S., Gorschek, T. (2015). The contextual nature of innovation - an empirical investigation of three software intensive products. *Information and Software Technology*, 57, pp. 595-613. <https://doi.org/10.1016/j.infsof.2014.06.010>

- Klotins, E. (2018). Software start-ups through an empirical lens: are start-ups snowflakes?. In 1st International Workshop on Software-Intensive Business: Start-Ups, Ecosystems and Platforms, SiBW 2018, Espoo, Finland, 3 December 2018. CEUR-WS.
- Klotins, E., Unterkalmsteiner, M., & Gorschek, T. (2018). Software engineering antipatterns in start-ups. *IEEE Software*, 36(2), 118-126.
- Klotins, E., Unterkalmsteiner, M., Gorschek, T. (2019). Software engineering in start-up companies: An analysis of 88 experience reports. *Empirical Software Engineering*, 24(1), 68-102.
- Kock, N. (2004). The three threats of action research: a discussion of methodological antidotes in the context of an information systems study. *Decision Support Systems*, 37, 265-286.
- Kon, Fabio and Cukier, Daniel and Melo, Claudia and Hazzan, Orit and Yuklea, Harry, A Panorama of the Israeli Software Startup Ecosystem (March 1, 2014). Available at SSRN: <https://ssrn.com/abstract=2441157> or <http://dx.doi.org/10.2139/ssrn.2441157>
- Kuhrmann, M. et al. (2021). What Makes Agile Software Development Agile. *IEEE Transactions on Software Engineering*. doi: 10.1109/TSE.2021.3099532.
- Langley, A. (1999). Strategies for Theorizing from Process Data. *Academy of Management Review*, 24(4).
- Lenarduzzi, V., & Taibi, D. (2016). MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product. In Proceedings of the 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).
- Linkner, J. (2017). *Hacking Innovation: The New Growth Model from the Sinister World of Hackers*. Fastpencil Publishing.
- Maine, E. (2008). Radical innovation through internal corporate venturing: Degussa's commercialization of nanomaterials. *R and D Management*, 38(4), 359-371.
- Maurya, A. (2012). *Running Lean: Iterate from Plan A to a Plan That Works*. O'Reilly Media Inc.
- McKelvie, A., Chandler, G., DeTienne, D., & Johansson, A. (2020). The measurement of effectuation: highlighting research tensions and opportunities for the future. *Small Business Economics*, 54(3), 689-720.
- Melegati, J., Goldman, A., Kon, F., & Wang, X. (2019). A model of requirements engineering in software startups. *Information and Software Technology*, 109, 92-107. <https://doi.org/10.1016/j.infsof.2019.02.001>
- Melegati, J., Guerra, E., & Wang, X. (2022). HyMap: eliciting hypotheses in early-stage software startups using cognitive mapping. *Information and Software Technology*, 144. doi: <https://doi.org/10.1016/j.infsof.2021.106807>.
- Morris, M. H., Kuratko, D. F., & Covin, J. G. (2010). *Corporate entrepreneurship & innovation*. Cen-gage Learning.

- Mullins, J., & Komisar, R. (2009). *Getting to Plan B: Breaking Through to a Better Business Model*. Harvard Business Review Press.
- Myers, M. D. (1997). Qualitative Research in Information Systems. *MIS Quarterly*, 21(2), pp. 241-242. MISQ Discovery, archival version, June 1997, <http://www.misq.org/supplements/>. Association for Information Systems (AISWorld) Section on Qualitative Research in Information Systems, updated version, last modified: May 12, 2021. [www.qual.auckland.ac.nz](http://www.qual.auckland.ac.nz)
- Myers, M., & Newman, M. (2007). The Qualitative Interview in IS Research: Examining the Craft. *Information and Organization*, 17(1), 2-26.
- Ng, P. (2015). Integrating software engineering theory and practice using essence: A case study. *Science of Computer Programming*, 101, pp. 66-78.
- Ng, P., & Huang, S. (2013). Essence: A framework to help bridge the gap between software engineering education and industry needs. In *Proceedings of the 2013 26th International Conference on Software Engineering Education and Training (CSEET)*, 2013, pp. 304-308, doi: 10.1109/CSEET.2013.6595266.
- Ng, P., Huang, S., & Wu, Y. (2013). On the value of essence to software engineering research: A preliminary study. 2013 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE).
- Nguyen-Duc, A., & Abrahamsson, P. (2016). Minimum Viable Product or Multiple Facet Product? The Role of MVP in Software Startups. In: Sharp H., Hall T. (eds) *Agile Processes, in Software Engineering, and Extreme Programming*. XP 2016. Lecture Notes in Business Information Processing, vol 251. Springer, Cham.
- Nguyen-Duc, A., Seppänen, P., & Abrahamsson, P. (2015). Hunter-gatherer cycle: a conceptual model of the evolution of software startups. In *Proceedings of the 2015 International Conference on Software and System Process (ICSSP)*, pp. 199-203.
- Nguyen-Duc, A, Shah, S. M. A., and Abrahamsson, P. (2016). Towards an Early Stage Software Startups Evolution Model. In *Proceedings of the 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 120-127, doi: 10.1109/SEAA.2016.21.
- Nguyen-Duc, A., Wang, X., & Abrahamsson, P. (2017) What influences the speed of prototyping? An empirical investigation of twenty software startups. In Baumeister, H., Lichter, H., & Reinisch, M. (eds) *Agile processes in software engineering and extreme programming lecture notes in business information processing*. Springer International Publishing, 20-36
- Object Management Group (2018). *Essence - Kernel and Language for Software Engineering Methods*. <https://www.omg.org/spec/Essence/1.0/PDF>
- Ojala, A. (2015). Business models and opportunity creation: How IT entrepreneurs create and develop business models under uncertainty, *26(5)*, 451-476.

- Ojala, A. (2016). Discovering and creating business opportunities for cloud services. *Journal of Systems and Software*, 113, 408–417
- Orlikowski, W., & Baroudi, J. (1991). Studying Information Technology in Organizations: Research Approaches and Assumptions. *Information Systems Research*, 2(1), 1-28.
- Osterwalder, A., Pigneur, Y., & Tucci, C. (2005). Clarifying business models: origins, present, and future of the concept. *Communications of the Association of Information Systems*, 15, 1–43.
- Pantiuchina, J., Mondini, M., Khanna, D., Wang, X., and Abrahamsson, P. (2017) Are Software Startups Applying Agile Practices? The State of the Practice from a Large Survey. In: Baumeister H., Lichter H., Riebisch M. (eds) *Agile Processes in Software Engineering and Extreme Programming. XP 2017. Lecture Notes in Business Information Processing*, vol 283. Springer, Cham. [https://doi.org/10.1007/978-3-319-57633-6\\_11](https://doi.org/10.1007/978-3-319-57633-6_11)
- Park, J. S. (2015). Essence-Based, Goal-Driven Adaptive Software Engineering. 2015 IEEE/ACM 4th SEMAT Workshop on a General Theory of Software Engineering.
- Park, J. S., McMahan, P. E., & Myburgh, B. (2016). Scrum Powered by Essence. *ACM SIGSOFT Software Engineering Notes*, 41(1), 1-8.
- Passaro, R., Rippa, P., & Quinto, I. (2016). The start-up lifecycle: an interpretative framework proposal. VII Annual Scientific Meeting of the Italian Association of Management Engineering (AiIG), - Higher Education and Socioeconomic Development, October 13-14, 2016, Bergamo, Italy, pp. 1-25.
- Patel, S., & Wormley, R. (2017). 100 Days of Growth Book - 100 Actionable Tips to Grow Your Startup Faster. E-Book. <https://100daysofgrowth.com/>
- Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., and Abrahamsson, P. (2014). Software Development in Startup Companies: a Systematic Mapping Study. *Information and Software Technology*, 56(10), pp. 1200-1218.
- Pieper, J., Lueth, O., Goedicke, M., & Forbrig, P. (2017). A case study of software engineering methods education supported by digital game-based learning: Applying the SEMAT Essence kernel in games and course projects. In *Proceedings of the 2017 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1689-1699. doi: 10.1109/EDUCON.2017.7943076.
- Pitchbook (2019). 4q 2018 PitchBook-NVCA venture monitor Pitchbook <https://pitchbook.com/news/reports/4Q2018-pitchbook-nvca-venture-monitor>
- Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional.
- Rasmussen, E. A., & Sørheim, R. (2006). Action-based entrepreneurship education. *Technovation* 26(2), 185-194.
- Reymen, I., Andries, P., Berends, H., Mauer, R., Stephan, U., & Burg, EV. (2015) Understanding dynamics of strategic decision making in venture creation:

- A process study of effectuation and causation. *Strategic Entrepreneurship Journal*, 9(4), 351–379.
- Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. New York: Crown Business.
- Rodríguez, P., Markkula, J., Oivo, M., & Turula, K. (2012). Survey on agile and lean usage in finnish software industry. *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 139-148.
- Runeson, P. & Höst, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, 14(2), p. 131.
- Salamzadeh, A., & Kawamorita K., H. (2015). Startup companies: Life cycle and challenges. In 4th International conference on employment, education and entrepreneurship (EEE), Belgrade.
- Sarasvathy, S. (2001). Causation and effectuation: Toward a theoretical shift from economic inevitability to entrepreneurial contingency. *Academy of Management Review*, 26(2), pp. 243–263.
- Seaman CB (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4), pp. 557–572.
- Seppänen, P. (2020). Yes, We Can! Building a Capable Initial Team for a Software Startup. In: Nguyen-Duc A., Münch J., Prikladnicki R., Wang X., Abrahamsson P. (eds) *Fundamentals of Software Startups*. Springer, Cham.
- Seppänen, P., Liukkunen, K., & Oivo, M. (2017). Little Big Team: Acquiring Human Capital in Software Startups. In *Proceedings of the 18th International Conference on Product-Focused Software Process Improvement, PROFES 2017, Innsbruck, Austria, November 29–December 1*, pp. 280-296.
- Seaman, C., & Guo, Y. (2011) Measuring and monitoring technical debt. *Advances in Computing*, 82, pp. 25–46
- SEMAT (n.d.). What is SEMAT? [semat.org/what-is-semat/](http://semat.org/what-is-semat/)
- SEMAT (2018). Great pick up of Semat. [http://semat.org/news/-/asset\\_publisher/eaHEtyeuE9wP/content/great-pick-up-of-semat](http://semat.org/news/-/asset_publisher/eaHEtyeuE9wP/content/great-pick-up-of-semat)
- SEVOCAB – Software and Systems Engineering Vocabulary (n.d.). [https://pascal.computer.org/sev\\_display/index.action](https://pascal.computer.org/sev_display/index.action)
- Sirelkhatim, F., and Gangi, Y. (2015). Entrepreneurship education: A systematic literature review of curricula contents and teaching methods. *Cogent Business & Management*, 2(1).
- Snowden, D., & Boone, M. (2007). A Leader's Framework for Decision Making. *Harvard Business Review*, Nov 2007, 69–76.
- Steinert, M., & Leifer, L. J. (2012). 'Finding One's Way': Re-Discovering a Hunter-Gatherer Model based on Wayfaring. *International Journal of Engineering Education*, 28(2), pp. 251-252.

- Susman, G. I., & Evered, R. D. (1978). An assessment of the scientific merits of action research. *Adm. Sci. Q.*, 582-603.
- Sutton, M. (2000). The Role of Process in a Software Start-Up. *IEEE Software*, 17(4), 33-39.
- Steininger, D. (2019). Linking information systems and entrepreneurship: A review and agenda for IT-associated and digital entrepreneurship research. *Information Systems Journal*, 29(2), pp. 363-407.
- The State of European Tech (2020). Retrieved from <https://2020.stateofeuropeantech.com>
- Tolvanen, J-P. (1998). Incremental Method Engineering with Modeling Tools - Theoretical Principles and Empirical Evidence. *Jyväskylä Studies in Computer Science, Economics and Statistics*, 47. [Doctoral Dissertation].
- Tripathi, N., Klotins, E., Prikladnicki, R., Oivo, M., Pompermaier, L. B., Kudakacheril, A. S., Unterkalmsteiner, M., Liukkunen, K., & Gorschek, T. (2018). An anatomy of requirements engineering in software startups using multivocal literature and case survey. *Journal of Systems and Software*, 146, 130-151. <https://doi.org/10.1016/j.jss.2018.08.059>
- Troisi, O., Maione, G., Grimaldi, M., & Loia, F. (2020). Growth hacking: Insights on data-driven decision-making from three firms. *Industrial Marketing Management*, 90, 538-557.
- Unterkalmsteiner, M., Abrahamsson, P., Wang, X., Nguyen-Duc, A., Shah, S., Bajwa, S., Baltés, G., Conboy, K., Cullina, E., Dennehy, D., Edison, H., Fernandez-Sanchez, C., Garbajosa, J., Gorschek, T., Klotins, E., Hokkanen, L., Kon, F., Lunesu, I., Marchesi, M., Morgan, L., Oivo, M., Selig, C., Seppanen, P., Sweetman, R., Tyrvaainen, P., Ungerer, C., & Yague, A. (2016). Software startups: a research agenda. *E-Informatica Software Engineering Journal*, 10(1), pp. 89-123. <https://doi.org/10.5277/e-Inf160105>
- Vakkuri V., Kemell K.-K., Abrahamsson P. (2019) Implementing Ethics in AI: Initial Results of an Industrial Multiple Case Study. In: Franch X., Männistö T., Martínez-Fernández S. (eds) *Product-Focused Software Process Improvement. PROFES 2019. Lecture Notes in Computer Science*, vol 11915. Springer, Cham. [https://doi.org/10.1007/978-3-030-35333-9\\_2](https://doi.org/10.1007/978-3-030-35333-9_2)
- Vakkuri, V., Kemell, K.-K., Abrahamsson, P. (2020). Internet resource for ECCOLA - a Method for Implementing Ethically Aligned AI Systems. figshare. Poster. <https://doi.org/10.6084/m9.figshare.12136308.v2>
- Vakkuri, V., Kemell, K.-K., Jantunen, M., Halme, E., and Abrahamsson, P. (2021). ECCOLA – A method for implementing ethically aligned AI systems. *Journal of Systems and Software*, 182.
- Vakkuri, V., Kemell, K.-K., Kultanen, J., & Abrahamsson, P. (2020). The Current State of Industrial Practice in Artificial Intelligence Ethics. *IEEE Software*, 37(4), 50-57. <https://doi.org/10.1109/MS.2020.2985621>
- Vakkuri, V., Kemell, K.-K., Kultanen, J., Siponen, M., & Abrahamsson, P. (2022). Ethically aligned design of autonomous systems: Industry viewpoint and an empirical study. To be published in the *Journal of Business Ethics and Organization Studies*, Summer 2022.

- Wang, G., & Nandhakumar, J. (2017). Strategic Swaying: How Startups Grow Digital Platforms. In Proceedings of the 2017 International Conference on Information Systems (ICIS).
- Wang, X., Edison, H., Bajwa, S. S., Giardino, C., & Abrahamsson, P. (2016, May). Key challenges in software startups across life cycle stages. In *International Conference on Agile Software Development* (pp. 169-182). Springer, Cham.
- Wohlin, C., & Aurum, A. (2015). Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, 20(6), 1427-1455. <https://doi.org/10.1007/s10664-014-9319-7>
- Yin, R. K. (2002). *Case Study Research, Design and Methods*, 3rd ed. Newbury Park, Sage Publications.
- Zachman, J. A. (2003) The Zachman framework for enterprise architecture Primer for Enterprise Engineering and Manufacturing. Zachman International.
- Zmееv, D. O., & Zmееv, O. A. (2020). Project-Oriented Course of Software Engineering Based on Essence. In Proceedings of the 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEET), pp. 1-3, doi: 10.1109/CSEET49119.2020.9206240.
- Zott, C., Amit, R., and Massa, L. (2011). The Business Model: Recent Developments and Future Research. *Journal of Management*, 37(4), 1019-1042.



## **APPENDIX: STARTUP CARDS FOR EARLY-STAGE STARTUPS**

This appendix features the method presented in Article V. As Article V has not yet been published at the time of the publication of this dissertation, the version of Article V included in this dissertation does not provide a way of properly accessing the method. The final version of Article V, once it is published, will contain some way of accessing the method. For the purposes of this dissertation, however, the method is included in this appendix as individual cards. There are 17 cards in total, and as such 17 cards are found in this appendix.

Customer  
Practice

## (1) Appealing Idea

**Motivation:** Without an idea there is no business. Business ideas are based on problems or needs. Existing businesses execute business models, startups look for them.

**What to do:** Choose a problem or need you wish to address. Plan a potential solution to tackle it.

If you do not already have an idea, start thinking about problems you personally face, or a need that you have that no current product can address. For additional inspiration, you may wish to look at new, emerging technologies, demographic or societal shifts, or simply watch other people online or offline.

**Common mistakes:** Falling in love with your idea and becoming unwilling to change it later on. Being afraid to share your own ideas.

**Sources:** [1] Alvarez, S. A., and Barney, J. B. (2007). Discovery and creation: alternative theories of entrepreneurial action. *Strategic Entrepreneurship Journal*, 1(1-2), 11-26.  
[2] Blank, S. (2013). *Four Steps to Epiphany*. K&S Ranch.  
[3] Ries, E. (2011). *The Lean Startup - How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Publishing.

Customer  
Practice

## (2) Great Pitch

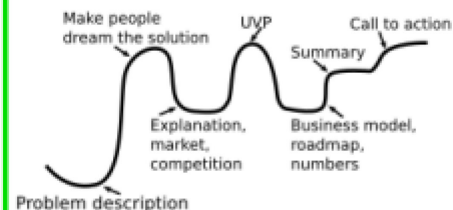
**Motivation:** Pitching means presenting your idea and your business in a concise manner. I.e., selling your idea.

**What to do:** Prepare a pitch for your startup and keep it up to date as you progress. Practice until you know it word-to-word. A pitch is usually 1 to 10 minutes. The focus of the pitch depends on your progress and your audience. Pitches typically include at least:

- (1) Problem: what problem are you solving?
- (2) Solution: how are you solving the problem?
- (3) Customer/User: who is your target audience?
- (4) Business: how do/will you make money? Numbers!
- (5) Market: how big is it? Is there competition?
- (6) Team: who are you?

Use evidence to back your claims up. Convince your audience you are not just selling them your hallucination.

**Common mistakes:** Unclear explanations. Failing to explain core aspects of your startup, e.g. business model. Not practicing your pitch beforehand.



**Sources:** [1] Colin Clark (2008) The impact of entrepreneurs' oral "pitch" presentation skills on business angels' initial screening investment decisions, *Venture Capital*, 10(3), pp. 257-279.  
[2] Moss, J. (2018). How to Make A Winning Pitch Deck for Your Startup. Retrieved from <<https://www.forbes.com/sites/forbesnyou/sci/2018/08/28/how-to-make-a-winning-pitch-deck-for-your-startup/>>

FIGURE 8. Startup Cards 1 and 2

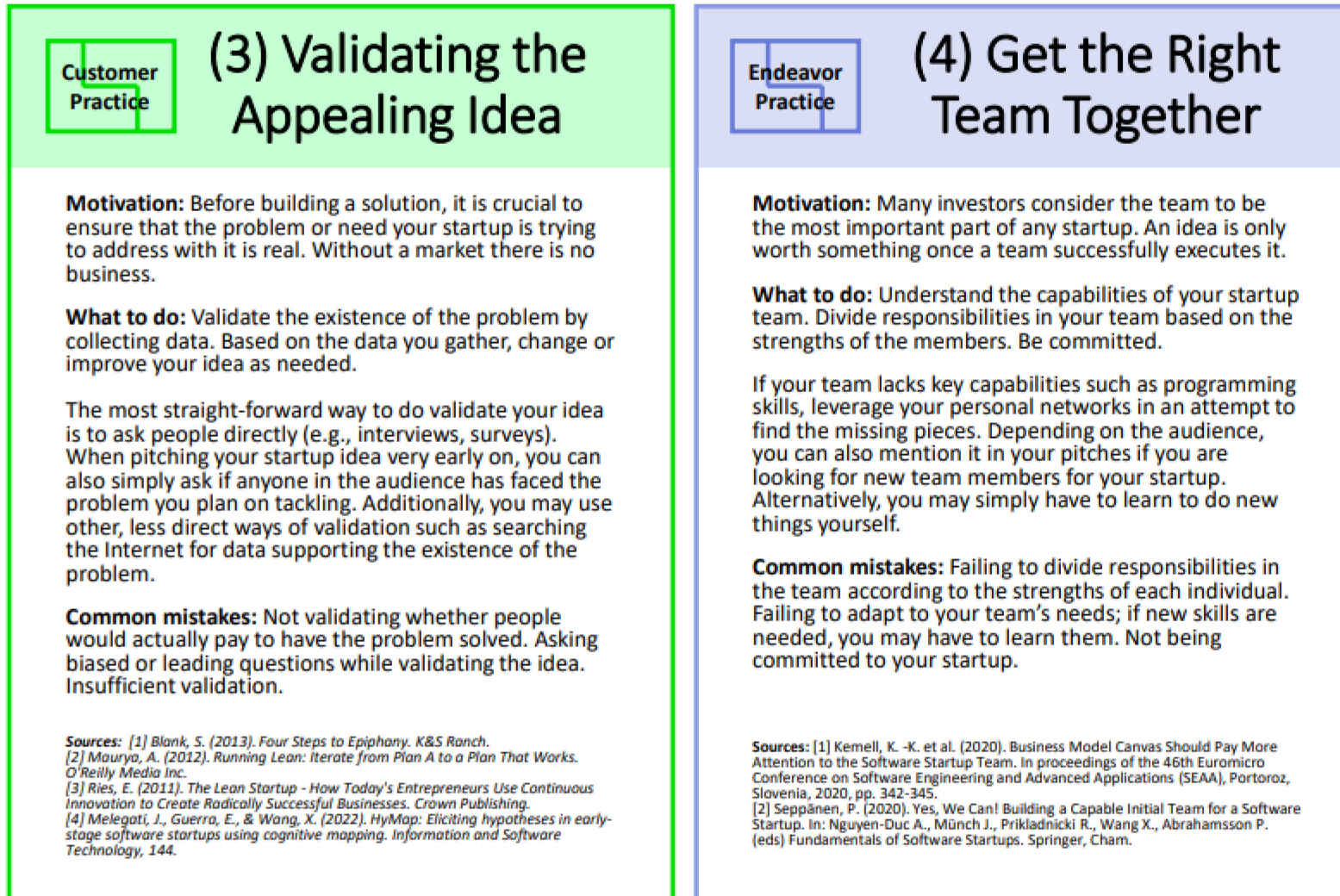


FIGURE 9. Startup Cards 3 and 4

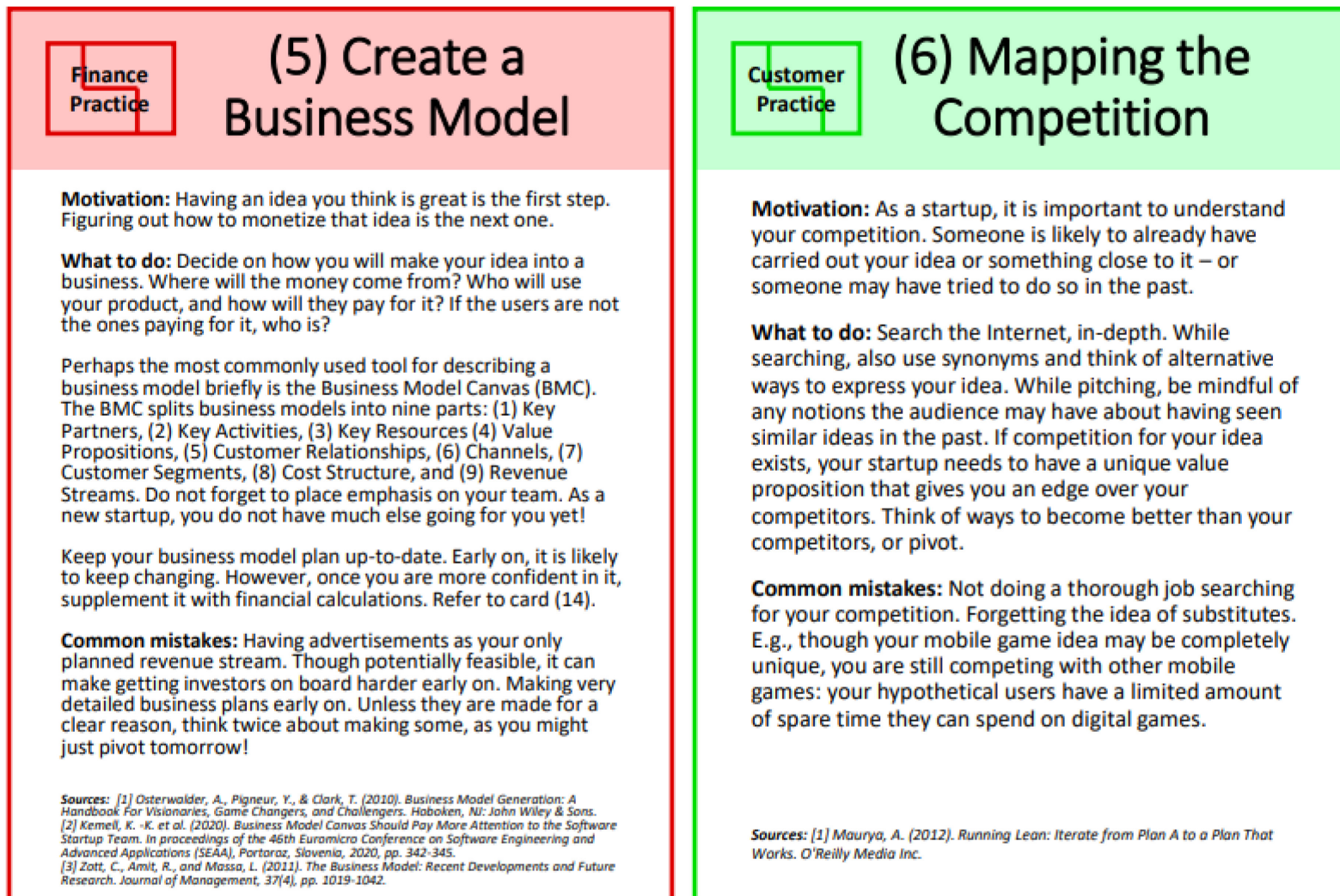
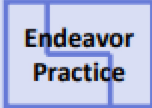


FIGURE 10. Startup Card 5 and 6


 Endeavor  
Practice

## (7) Establish Your Initial Way-of-Working


**Motivation:** To work efficiently, your startup team has to agree on a way-of-working. A way-of-working encompasses work from team communications to programming.

**What to do:** Agree on the tools and methods your startup team will use. How will you communicate? When and where will you work? What software will you use?

Your way-of-working should evolve as you progress. Practices and methods should be changed based on what is the most effective in any given situation. As you work, reflect on what you are doing and ask yourselves if you could be doing things better.

**Common mistakes:** Failing to reflect on your way-of-working in order to improve it. Working in an unsystematic fashion.

**Sources:** [1] Jacobson, I., Ng, P., McMahon, P. E., Spence, I., and Lidman, S. (2012). *The Essence of Software Engineering: The SEMAT Kernel*. *ACMQueue*, 10, pp. 40-52.  
[2] Kemell, K.-K., Ravaska, V., Nguyen-Duc, A., & Abrahamsson, P. (2020). *Software Startup Practices: Software Development in Startups Through the Lens of the Essence Theory of Software Engineering*. In *Proceedings of the 21st International Conference on Product-Focused Software Process Improvement*, pp. 402-418.


 Customer  
Practice

## (8) Validating the Potential Solution

**Motivation:** Once you have validated that the need or problem your startup is trying to address is a real one people are willing to pay to have addressed, the solution needs to be validated. Your solution might still not be the right one for addressing it.

**What to do:** Gather data. Use different means of data collection to understand whether people 1) are interested in your solution, and 2) willing to pay for it. Combine different data to gain better insights.

At the early stages, surveys and interviews can help with validation. However, iteratively working with Minimum Viable Products is a great way to validate your solution. Other means of data collection for gauging interest include social media, setting up a website for the company etc.

Ultimately, though, the best validation is having users or having pre-orders for your solution.

**Common mistakes:** Falling in love with your idea and being afraid to change it based on the data. Asking leading questions. Failing to consider that when confronted face-to-face, people are likely to agree with you simply to avoid conflict and to please you. Failing to understand the needs of the potential users or customers.

**Sources:** [1] Nguyen-Duc, A., and Abrahamsson, P. *Minimum Viable Product or Multiple Facet Product? The Role of MVP in Software Startups*. In: Sharp H., Hall T. (eds) *Agile Processes, in Software Engineering, and Extreme Programming. XP 2016. Lecture Notes in Business Information Processing*, vol 251. Springer, Cham  
EuroMicro Conference on Software Engineering and Advanced Applications (SEAA).  
[2] Blank, S. (2013). *Four Steps to Epiphany*. K&S Ranch.  
[3] Melegati, J., Guerra, E., & Wang, X. (2022). *HyMap: Eliciting hypotheses in early-stage software startups using cognitive mapping*. *Information and Software Technology*, 144.

FIGURE 11. Startup Cards 7 and 8

**Solution  
Practice**

## (9) Be Ready to Pivot

**Motivation:** The act of changing direction, business-wise; a strategic decision. Knowing when to pivot or to persevere is crucial for a startup. Few ideas are perfect from their inception; e.g., YouTube was originally a video dating service.

**What to do:** Use data to support decision-making. Tracking metrics provides valuable data on when to pivot. What metrics to track depends on your progress and your business. For example, early on, if surveying potential users indicates that people simply are not interested in your idea, do not hesitate to change it.

Pivots can be made in relation to any aspect of your startup. For example, customer need pivots change the core problem being solved, while customer segment pivots change the target users or customers.

**Common mistakes:** Failing to pivot despite data pointing towards a need for a pivot. Failing to track relevant metrics. Blindly believing that your idea will find its market once it is finished.

**Sources:** [1] Bajwa, S. S., Wang, X., Nguyen-Duc, A., and Abrahamsson, P. (2016). How Do Software Startups Pivot? Empirical Results from a Multiple Case Study. In: Maglyas A., Lamprecht AL, (eds) Software Business. ICSSB 2016. Lecture Notes in Business Information Processing, vol 240. Springer, Cham.  
[2] Bajwa, S., Wang, X., Nguyen-Duc, A., Chanin, R., Prikladnicki, R., Pompermaier, L., & Abrahamsson, P. (2017). Start-Ups Must Be Ready to Pivot. *IEEE Software*, 34, 18-22.  
[3] Bajwa, S., Wang, X., Nguyen-Duc, A., & Abrahamsson, P. (2017). "Failures" to be celebrated: an analysis of major pivots of software startups. *Empirical Software Engineering*, 22(5).

**Customer  
Practice**

## (10) Utilize Metrics

**Motivation:** For a startup, metrics are performance indicators. Utilizing metrics can help you evaluate your startup's current performance and the successfulness of your idea or solution. They help you make informed decisions instead of relying on intuition alone.

**What to do:** (1) Collect data, even if you have no clear use for it right now. Hard-drive space is cheap. Later on, you can analyze the data to discover valuable lessons. (2) Identify key metrics for the current state your startup is in. For example, if your service is a game, the most important metric for you might be whether your players perform a specific action in the game, and how often. This might predict user retention/churn etc.

Combine metrics for more insights. For example, comparing Daily Active Users (DAU) and Monthly Active Users (MAU) gives you a better understanding of your user activity than either metric alone.

**Common mistakes:** Not collecting data. Not combining different types of data. Having no goal for your metrics.

**Sources:** [1] Kemell, K.-K., Wang, X., Nguyen-Duc, A., Grendus, J., Tuunanen, T., & Abrahamsson, P. (2018). 100+ Metrics for Software Startups: A Multi-Vocal Literature Review. In S. Hyrynsalmi, A. Suominen, C. Jud, X. Wang, J. Bosch, & J. Münch (Eds.), *SIBW 2018: Proceedings of the First International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms*, pp. 15-29.

FIGURE 12. Startup Cards 9 and 10

**Solution  
Practice**

## (11) Minimum Viable Product

**Motivation:** The idea of a Minimum Viable Product (MVP) is to launch your product, in some form, as early as possible in order to further validate and improve your business idea. Each MVP should have a goal.

**What to do:** Forget the idea of prototypes that are nearly finished products. MVPs can be many things and should be used as early as possible! Build your first MVP in a day or two and get learning. Collect data and use it to validate something.

An MVP is not just about users being able to use your solution in practice. An MVP can also be something that convinces potential future users to pre-order your solution or to at least track your startup's progress through, e.g., an email list. A simple early MVP can be an ad on Google, a landing page, or a mock-up software application.

More complex MVPs, such as functional prototypes, only come later. Be creative with MVPs and work iteratively: use the data to improve your idea.

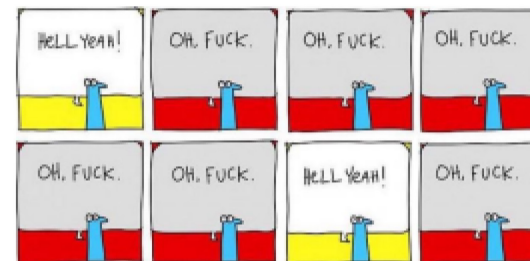
**Common mistakes:** Defining your MVP to be an almost finished solution. Forgoing an MVP altogether while working on a supposedly definitive solution. Having no goal in mind for your MVPs.

**Sources:** [1] Nguyen-Duc, A., and Abrahamsson, P. Minimum Viable Product or Multiple Facet Product? The Role of MVP in Software Startups. In: Sharp H., Hall T. (eds) Agile Processes, in Software Engineering, and Extreme Programming. XP 2016. Lecture Notes in Business Information Processing, vol 251. Springer, Cham  
[2] Lenarduzzi, V., and Taibi, D. (2016). MVP Explained: A Systematic Mapping Study on the Definitions of Minimal Viable Product. In Proceedings of the 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).  
[3] Blank, S. (2013). Four Steps to Epiphany. K&S Ranch.

**Endeavor  
Practice**

## (12) Startup Spirit

- **Motivation:** Being a startupper is almost never smooth sailing. However, it's only truly over once you stop believing in your own idea.
- **What to do:** Get into the mindset of being a startup founder. Learn from your experiences, especially the negative ones, and do not get disheartened by pivots. Forget counting hours spent working; no one will pay you for them until you can do so yourself or get funded.
- Tasks are done when you decide they are done, not when you have spent n hours on them. You are your own boss. Manage your own time well.
- **Common mistakes:** Treating your startup as just a course project and not a real startup (in the context of this course). Losing motivation in the face of setbacks.



from @tobiasstraka (<https://twitter.com/tobiasstraka/status/750399141260460033>)

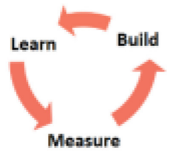
FIGURE 13. Startup Cards 11 and 12

**Solution  
Practice**

## (13) The Learn-Measure- Build Loop

(a.k.a. Build-Measure-Learn)

**Motivation:** Learning is vital for a startup, and in order to learn you need data to learn from.



**What to do:** Build MVPs. An MVP is the core of the Learn-Measure-Build loop. By using an MVP you can gather data that helps you learn.

Before you start building an MVP, plan. Do not build without having an objective. First, you need to decide what you want to learn. Then, you plan on how you can learn it. What is the MVP that will answer your question(s) best, and with the least amount of time wasted? I.e., what data do you collect and how?

The process is iterative. Test different hypotheses and seek to answer different questions. Between each MVP, leverage your lessons learned from the previous one.

**Common mistakes:** Wasting time on far too complex MVPs. Building MVPs without planning and without setting goals. Failing to iterate the process.

**Sources:** [1] Blank, S. (2013). *Four Steps to Epiphany*. K&S Ranch.  
[2] Ries, E. (2011). *The Lean Startup - How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Publishing.  
[3] Bosch, J., Holmström Olsson, H., Björk, J., & Ljungblad, J. (2013) *The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups*. In *Proceedings of the 2013 International Conference on Lean Enterprise Software and Systems (LESS2013)*. Lecture Notes in Business Information Processing, vol 167.

**Finance  
Practice**

## (14) Calculate the Financial Metrics

**Motivation:** Even if you have a compelling idea for a business, potential investors will want to know what exactly you need money for, and how you will make it back.

**What to do:** After deciding on your business model, start calculating financial metrics for your startup. In the early stages, these are largely hypothetical future forecasts. However, forecasts help you better understand the costs of your business and what is thus needed to profit. You cannot price your services reliably if you do not understand your startup's cost structure.

Break your costs and revenue streams down into parts: what will cost you money and what will bring you money? Understand which costs are fixed and which are variable; variable costs scale with your startup. Compare your costs with your current or hypothetical future revenue streams. When you know your costs, you know what exactly your startup needs funding for, and how much.

As you progress, keep updating these forecasted metrics with real data. Financial metrics are very interesting to potential investors and thus relevant in any pitch past the very early stages of merely pitching your general idea.

**Common mistakes:** Not putting in the effort to make detailed calculations; making lazy and unrealistic forecasts. Thinking a general description of your business model alone is enough to convince investors.

**Sources:** [1] Moss, J. (2018). *How to Make A Winning Pitch Deck for Your Startup*. Retrieved from <<https://www.forbes.com/sites/forbesyncouncil/2018/08/28/how-to-make-a-winning-pitch-deck-for-your-startup/>>. 29 Oct 2018.  
[2] Zott, C., Amit, R., and Massa, L. (2011). *The Business Model: Recent Developments and Future Research*. *Journal of Management*, 37(4), pp. 1019-1042.

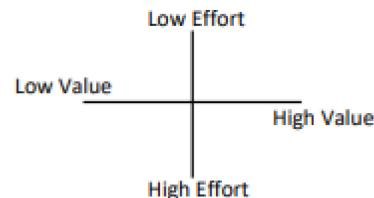
FIGURE 14. Startup Cards 13 and 14



**Solution  
Practice**

## (15) Manage Scope

**Motivation:** Adding new features takes resources, and there is *always* more you could add. Done is better than perfect, as the anecdotal wisdom goes.



**What to do:** Determine the core value of your service and focus on it. Why will people use your service? What does it *need* to do for your users to use it? Separate the necessities from the niceties (“nice to have”).

Requirements and features need to be prioritized. Weigh them based on how much resources they take to implement and how much value they deliver. Focus on the ones that result in the biggest impact with the least resources spent. Leave the ‘niceties’ for when you already have a functional product. The matrix on the upper right is one way of prioritizing them.

**Common mistakes:** Constantly pivoting past the earlier stages of initial idea validation. Adding every new idea to the product backlog. Continuing to delay service launch to add more and more features past the essential ones.

**Sources:** [1] Klotins, E., Unterkalmsteiner, M., & Garschek, T. (2019). *Software Engineering Antipatterns in Start-Ups*. *IEEE Software*, 36(2), pp. 118-126.  
[2] Klotins, E., Unterkalmsteiner, M., & Garschek, T. (2019). *Software engineering in start-up companies: An analysis of 88 experience reports*. *Empirical Software Engineering*, 24, 68-102.

**Customer  
Practice**

## (16) Work With Your (Future) Users

**Motivation:** Having users, or having people who wish to start using your service once you release it, already validates your idea. However, these users and future users provide you with a great way of further validating and improving your solution.

**What to do:** If you already have a functional service, collect data from it. Decide on metrics and track them. See the *Utilize Metrics* card.

Actively engage with your users or future users. You can get in direct contact with your first (end-)user(s), especially if you are a B2B startup. You can directly ask for feedback through interviews, or by less formal means, and then use this data to design new features or improve existing ones. Work with them, if they want to do so.

As a B2C startup, staying in contact with many users directly can be daunting. Instead, by using an existing platform (e.g., Discord), you can create a community where your users and future users can talk to you about the service. Vice versa, you can also communicate with your users about future updates and planned features to collect feedback.

**Common mistakes:** Failing to differentiate between *wants* and *needs*. Listening too much to one user or one group of users. Providing too much customization to single users (e.g., B2B).

**Sources:** [1] Klotins, E., Unterkalmsteiner, M., & Garschek, T. (2019). *Software Engineering Antipatterns in Start-Ups*. *IEEE Software*, 36(2), pp. 118-126.  
[2] Ries, E. (2011). *The Lean Startup - How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Publishing.

FIGURE 15. Startup Cards 15 and 16

## (17) Make it Stable

**Motivation:** Nothing is more frustrating than tech that simply does not work. Crashes and slow loading times can drive off users incredibly fast. Your service might still lack various features you plan to implement later, but it should not lack basic quality if you want to deploy it.

**What to do:** Test, test, and test. Test internally, ask your friends and family to test it, or pay external testers to do so. Test on different hardware and platforms. Try different types of testing.

Even if you are talking about an MVP that is an early, functional version of your service, it should still be stable above all. Losing users to simple quality issues means you are not seeing what *else* might have driven them away. Quality issues are not service-specific while your other issues might be!

**Common mistakes:** Forgoing all quality control in favor of speed. Failing to detect existing quality issues during deployment.

Sources: [1] Klatins, E., Unterkalmsteiner, M., & Garschek, T. (2019). Software Engineering Antipatterns in Start-Ups. *IEEE Software*, 36(2), pp. 118-126.

FIGURE 16. Startup Card 17



## ORIGINAL PAPERS

### I

# THE ESSENCE THEORY OF SOFTWARE ENGINEERING: LARGE-SCALE CLASSROOM EXPERIENCES FROM 450+ SOFTWARE ENGINEERING BSC STUDENTS

by

Kai-Kristian Kemell, Anh Nguyen-Duc, Xiaofeng Wang, Juhani Risku & Pekka  
Abrahamsson, 2018

*19th International Conference on Product-Focused Software Process Improvement*  
(PROFES) (pp. 123-138). Springer. Lecture Notes in Computer Science, 11271

DOI 10.1007/978-3-030-03673-7\_9

Reproduced with kind permission by Springer.

# The Essence Theory of Software Engineering – Large-Scale Classroom Experiences from 450+ Software Engineering BSc Students

Kai-Kristian Kemell<sup>1</sup>[0000-0002-0225-4560] Anh Nguyen-Duc<sup>2</sup>[0000-0002-7063-9200] Xiaofeng  
Wang<sup>3</sup>[N/A] Juhani Risku<sup>1</sup>[N/A] and Pekka Abrahamsson<sup>1</sup>[0000-0002-4360-2226]

<sup>1</sup> University of Jyväskylä, 40014 Jyväskylä, Finland  
{kai-kristian.o.kemell|pekka.abrahamsson|juhani.risku}@jyu.fi

<sup>2</sup> University of Southeast Norway  
angu@usn.no

<sup>3</sup> Free University of Bozen-Bolzano, 39100 Bozen-Bolzano, Italy,  
xiaofeng.wang@unibz.it

**Abstract.** Software Engineering as an industry is highly diverse in terms of development methods and practices. Practitioners employ a myriad of methods and tend to further tailor them by e.g. omitting some practices or rules. This diversity in development methods poses a challenge for software engineering education, creating a gap between education and industry. General theories such as the Essence Theory of Software Engineering can help bridge this gap by presenting software engineering students with higher-level frameworks upon which to build an understanding of software engineering methods and practical project work. In this paper, we study Essence in an educational setting to evaluate its usefulness for software engineering students while also investigating barriers to its adoption in this context. To this end, we observe 102 student teams utilize Essence in practical software engineering projects during a semester long, project-based course.

**Keywords:** Software Engineering, Method, Practice, Essence, SEMAT, Education, Software Process Engineering

## 1 Introduction

Software Engineering (SE) work out in the field is diverse, with practitioners employing a myriad of different methods and practices in equally diverse SE endeavors [5, 10]. As little consensus exists in terms of best practices and methods, practitioners have taken to using what they consider to be the best option(s) for their own SE context, often tailoring them by omitting some suggested practices or rules [5]. Though e.g. Agile methods are currently widely employed out on the field, the practices and methods that are understood as being Agile are numerous [1]. Especially software startups use a diverse mix of agile methods and practices, with some simply opting to use ad hoc SE methods [17].

This diversity in the SE industry has, alongside other factors such as technological advances, resulted in a gap between education and practice in SE [2, 13]. As it is not possible to teach university students all the methods and practices employed by

practitioners, curriculum-makers are faced with choices on what to focus on. General theories and methods that can be taught to students to support them in the adoption of new practices in the future are one option in attempting to tackle this gap. One such theory is the *Essence Theory of Software Engineering* (Essence from here-on-out), proposed by the SEMAT initiative<sup>1</sup> [10].

Created to address the vast range of methods employed in the field, Essence is a method-agnostic progress control tool for SE. Essence is modular in nature and can be used to model any existing methods, practices, or combination of such [15]. Thus, Essence is designed to suit any SE possible context [9], making it a potentially powerful tool. However, its flexibility is also a potential downside: in order to use Essence, resources have to be devoted towards modeling the practices and methods being used, as well as learning how to do specifically by using Essence.

Presently, Essence has yet to see widespread adoption among practitioners, although it has seen some traction among the academia [21]. It is possible that its rather resource-intensive adoption is one barrier for its adoption, as has been discussed in extant research [8, 18]. For this purpose, some tools have been suggested to aid practitioners in its adoption and in using it: e.g. [8] presented SematAcc to help users visually track the alpha states while using Essence and [11] presented an Essence-themed board game to make learning Essence easier. However, more tools and further studies specifically focusing on its supposedly difficult adoption are also required to better understand the barriers of its adoption and to consequently be able to tackle them. Additionally, an educational perspective on Essence is interesting because Essence can help address the gap between education and industry needs. For example, [2] report that SE graduates are often perceived by the industry as lacking in e.g. the ability to follow processes and project management skills, both of which Essence can help teach.

In this paper, we study Essence in a large-scale classroom setting. We observe over one hundred project teams consisting of second year SE students employ Essence during course projects mimicking a field SE endeavor. The teams carry out a complete SE project, from requirements formulation to a finished software product, using Essence to manage their project. Then, based on their projects, the students reflect on their experiences with Essence in a written experience report. With the data collected from these experience reports, we seek to understand:

**RQ1:** *How useful do bachelor level students find Essence?*

**RQ2:** *What are the challenges in adopting Essence, specifically for inexperienced software developers, and what could be done to make its adoption easier?*

The rest of this paper is structured as follows. In the next section, we discuss the Essence specification and extant research on it in further detail. In the third section, we present and discuss the study design. In the fourth section, we analyze the data and present our findings. We then discuss the practical and theoretical implications of our findings in the fifth section, as well as the potential limitations of the study and directions for future research. The sixth and final section concludes the paper.

---

<sup>1</sup> semat.org

## 2 The Essence Theory of Software Engineering

Essence is a modular, method-agnostic progress control tool for SE endeavors. Proposed by the SEMAT initiative to address the myriad of methods and practices employed by industry practitioners, Essence is a framework into which any combination of existing methods or practices can be inserted. In practice, Essence consists of a kernel and a language. The kernel [14], its authors argue [10], contains all the elements present in every SE endeavor, while the language can be used to extend the kernel to fit any specific SE endeavor. I.e. Essence, in its base form, contains the elements required to track progress in a generic SE endeavor, but it is intended to be tailored for specific SE contexts.

The Essence kernel consists of three views: *alphas*, *activity spaces*, and *competencies*. In the kernel, there are seven alphas (Fig. 1.), “things to work with”: opportunity, stakeholders, requirements, software system, work, team, and way of working [10]. These alphas, Jacobson et al. [10] posit, are present in every SE endeavor. Alpha is an acronym for an “Abstract-Level Progress Health Attribute” [14]. For the project to progress, these alphas need to be worked on. To this end, the kernel contains activity spaces. Activity spaces may contain 0 or n activities, or “things to do”. The activity spaces in the kernel, much like the alphas, are elements Jacobson et al. [10] argue are found in every SE endeavor. Finally, the kernel contains a set of competencies: skills needed to carry out the endeavor [10]. These alphas, activity spaces, and competencies are further split into three areas of concern: *endeavor*, *solution*, and *customer*.

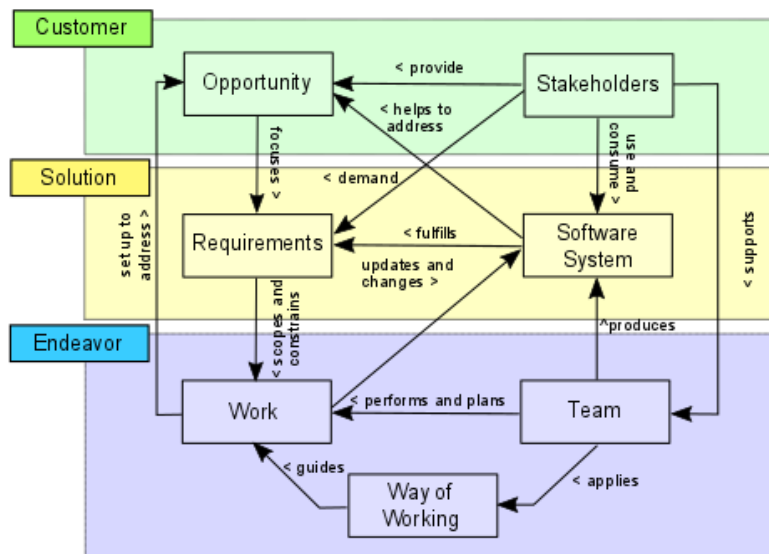


Fig. 1. The Essence Kernel Alphas

The alphas of the kernel serve as a way of tracking project health. *Alpha states* offer a way of tracking progress on the various areas of the endeavor. Each of the seven base alphas has a set of states that describe the progress made on each individual alpha. For

example, the states for the requirements alpha range from conceived, where the requirements have only just been formulated, to fulfilled, where they have been implemented into the system in a manner satisfying the stakeholders.

Jacobson, Stimson & Hastie [9] suggest Essence as a solution to what they call method prisons. In speaking of method prisons, they refer to the idea of organizations being stuck following one method or set of methods regardless of their suitability in the current context at any given time. However, they posit, the SE practitioners often present methods as monolithic for example by using very varied presentation styles to describe them. By presenting methods in a uniform manner, by e.g. using Essence, and by simply promoting a method-agnostic idea, Jacobson et al. [9] argue that organizations could escape method prisons and potentially improve their work processes by creating better methods specifically suited for their SE context.

Though its modular and extensible nature is the greatest strength of Essence, it can also be its greatest weakness. Whereas it makes Essence a powerful tool, it also makes it both resource-intensive and potentially difficult to adopt. Perhaps consequently, Essence has not gained widespread recognition among practitioners, although it has gained some traction among the academia [21]. Graziotin & Abrahamsson [8] suggest that the modest attention Essence has received among practitioners may well stem from the steep learning curve of the specification. Even though Jacobson et al. [9] make a potentially interesting case in promoting the idea of tailoring methods more actively, it may seem easier for practitioners to get started by simply using an existing method.

### 3 Research Design and Methodology

In this section, we describe the methodology of the classroom study on Essence in the context of student SE projects. In the first sub-section, we discuss the course from which the data was collected. The role of Essence in said course is then discussed in the second sub-section. The third and final sub-section discusses our data collection and analysis methodology in detail. The data is then analyzed in the following main section.

#### 3.1 The Course

The study presented in this paper was conducted using data from the *TDT4140 – Software Engineering* course at the Norwegian University of Science and Technology (NTNU). More specifically, all data for this study was collected during the 2017 spring iteration of the course during which the students utilized Essence in their projects. In this instance of the course, each project team was to engineer a functional software by carrying out a real SE project in a university setting. The theme of the projects was to radically improve university education by means of software robots. The exact goal of the projects was to “make a bot to replace Prof. Abrahamsson at his course on SE”.

Following the first lecture of the course, the students were instructed to form project teams consisting of 4 to 5 students. The teams were formed by having the students give a subjective evaluation of their own programming skills in terms of programming confidence and then form teams with individuals with similar evaluations. This was done to negate any potential internal issues (e.g. workload distribution issues) within the

teams arising from skill differences in programming. Starting from the first lecture, these teams were to work on their projects until the end of the course. The teams were first tasked with interviewing university teaching staff in order to discover tangible needs that could be addressed through their software. Stakeholders were involved in this fashion to make the project mimic a real SE endeavor more closely.

After gathering needs through the interviews and selecting the one(s) they wished to address, the students were to plan their development methodology and start utilizing it. During the course and the projects, weekly two-hour-lectures continued to offer relevant information and to support the project teams. The project work itself was carried out largely independently by each team.

### 3.2 The Role of Essence in the Course

Essence was introduced to the teams in the first lecture. The first lecture focused on discussing SE work in practice, specifically from the point of view of projects. During the lecture, Essence was discussed primarily in relation to its seven alphas, which were underlined to present the essential elements of an SE endeavor. In terms of methods, the students were instructed to initially work in whatever fashion they thought was best. The reasoning behind this line of action was to create fertile ground for the later adoption of Essence: by letting the teams first work in a rather unsystematic or even ad hoc fashion, they would likely be more receptive to tools that could help them systematize their way of working. I.e. having experienced unsystematic SE project work, they would better understand the need for more structured approaches to SE.

This approach, in practice, resulted in the teams largely working with various “ScrumBut”<sup>2</sup> approaches for the first three weeks. Their use of Scrum was likely to have stemmed from a previous course at the university having introduced them to Scrum. After three weeks of working as they saw fit without outside assistance from the teaching team, the teams were introduced to the Ivar Jacobson Practice Library<sup>3</sup>. They were tasked with using the practice cards (Fig. 2) from the library to re-construct their way of working and to modify it as they saw fit based on their experiences so far.

In this fashion, the teams were introduced to both the progress control aspect of Essence and its method-agnostic philosophy during the course. After the introduction of the practice cards, the use of Essence was not enforced during the project work and there were no regular check-ups to confirm its utilization. Full and correct utilization of Essence was not mandatory, and its utilization or lack thereof did not affect the grades given to the teams. All teams were instructed to utilize it to what extent they felt they could, but this was not supervised in practice. This approach was chosen to gather more unbiased data on the possible barriers of adoption in the case of Essence.

---

<sup>2</sup> ScrumBut refers to using Scrum while omitting some parts of it, “We use Scrum, but...” (refer to: <https://www.scrum.org/resources/what-scrumbut>)

<sup>3</sup> <https://practicelibrary.ivarjacobson.com/start>





**Fig. 2.** A project team showing their practice cards

### 3.3 Data Collection and Analysis Methodology

The data for this study was collected through written reflective reports provided by each team at the end of their projects<sup>4</sup>. In their report, each team was instructed to reflect on their experiences with Essence, along with other content unrelated to this study. As for Essence, they were to describe how they utilized it and how they felt about having done so. More specifically: (1) what they thought was good about Essence, (2) what they thought was bad about Essence, and (3) how they utilized Essence during their project.

Ultimately, 102 project teams of 4-5 students finished the course and delivered a written project report. Our data analysis is based on these 102 reports. The teams were not given a strict format to follow in the sections of their reports describing Essence, which led to the data being somewhat diverse in presentation. Each report was to discuss the afore-mentioned three topics related to their use of Essence, but past these general guidelines the Essence sections of the reports were freeform. In practice, this largely just meant that teams that had utilized Essence relatively little wrote little about it whereas teams that had utilized it fully wrote far more about their experiences.

Thematic analysis was chosen as the method of analysis for this study due to the large volume of the data, as well as the lack of pre-determined assumptions of how the students possibly perceived the use of Essence in this context. Both the final themes and the initial codes used to formulate them were generated from the data in an inductive fashion. The analysis process was iterative and reflexive.

Initially, the author conducting the thematic analysis went through the data and recorded key points for each report, both by directly quoting the reports and by making summarizing remarks, in a separate text document. During this process, initial codes were formulated based on recurring sentiments in the reports. E.g. many reports turned

---

<sup>4</sup> A book showcasing the results of the projects can be found on Figshare: [https://figshare.com/articles/100\\_Open\\_Sourced\\_Software\\_Robots\\_for\\_Tomorrow\\_s\\_Education\\_Revolutionizing\\_the\\_University\\_Learning\\_Experience\\_with\\_Bot\\_Technologies/5597983](https://figshare.com/articles/100_Open_Sourced_Software_Robots_for_Tomorrow_s_Education_Revolutionizing_the_University_Learning_Experience_with_Bot_Technologies/5597983)

out to describe various initial difficulties in adopting Essence. The analysis process was iterative, and reports and the recorded key points and quotations were regularly re-read as further codes were generated. This phase was concluded once all reports had been analyzed and the final set of codes had been applied to each of them where applicable.

Finally, the themes were generated inductively from the coded data. Codes were arranged into matching themes, with each theme encompassing one or more codes. In determining the themes, the research questions were used as a framework for organizing the data under the themes as well as determining the relevance of the codes and what was to ultimately be included into the study. In presenting the results in the next section, some of the direct quotations used in the analysis process were also included.

Additionally, in our first research question we speak of *usefulness*. Usefulness is a construct often used in relation to evaluating software systems designed especially for work-related use (e.g. [4]). In the context of this study, we define usefulness to be related to either learning something new about SE or SE progress control (educational usefulness) or providing help in SE project work (practical usefulness). These two seemingly separate types of usefulness are nonetheless closely linked together, however. E.g. a learning experience related to SE project work may simultaneously result in practical usefulness through the application of the newly-learned information into practice, which may also take place at a later point in time. In our analysis, we thus speak of usefulness while referring to usefulness in both senses.

## 4 Results

The reports showed a very varying degrees and success of utilization of Essence among the 102 project teams. Whereas some of the teams had clearly utilized Essence in its entirety and reflected upon it in depth, some of the teams had done the bare minimum of selecting different practices to use while forgoing the progress control aspect of Essence. However, despite the varying degree and success of Essence utilization among the teams, the reports discussed similar themes across the spectrum.

### 4.1 Theme 1: Difficult or Resource-Intensive to Learn

The reports indicated that the majority of the teams considered Essence difficult to learn to some extent. Even most of the teams that ultimately utilized Essence successfully considered it to have been difficult to initially grasp. As the course involved only a general introduction to Essence and its principles, the teams were to study and use Essence on their own using what resources they would find on the SEMAT website or the Internet in general. This resulted in most teams feeling that Essence was difficult to learn, or “hard to get a grasp on when first introduced” (Report 048). The teams generally considered to be a direct result of the types of resources available online:

*...we felt that almost anywhere we went to read about SEMAT we were either drowned with information (the Essence Kernel PDF has 308 pages) or the information was too abstract that we felt left confused after reading. (Report 041)*

*The web page material, the articles and the academic resources about SEMAT are filled with many new terms, but few clear definitions. It would be easier for the next years students to grasp what SEMAT really is, if there existed some sort of document on blackboard explaining the SEMAT terminology. (Report 016)*

Largely in line with the quotation above, though Essence was considered difficult to learn, the teams almost uniformly cited the lack of good tutorial resources as the main reason for this. The existing ones were considered either too lengthy or too simply written in a needlessly complex manner, failing to offer a good initial touch to the specification. This is also supported by some reports directly stating that past the initial barrier of adoption, Essence was a useful tool. However, due to its resource-intensive adoption, many felt that they wanted to focus on the practical SE work instead:

*We just wanted to get on with the programming and it seemed like it was just one more unnecessary thing we needed put effort into when we already had quite a lot with learning new technologies and languages. (Report 044)*

Past the self-reported issues related to learning Essence, it was also occasionally possible to determine that a team had not managed to internalize Essence based on the contents of their report. It was evident that some teams had only utilized the practice cards, as they had been directly instructed to do, and ignored the kernel and its alphas and other views, i.e. the progress control aspect of Essence. It is likely that this was caused by the perceived difficulty of learning the specification: some of these teams likely felt that they had understood Essence despite only grasping parts of it. Though the difficulty of learning Essence was primarily blamed on the lack of good tutorial resources, one of the teams did specifically state that they felt Essence itself was too abstract for them.

Despite Essence being considered somewhat difficult to initially learn by the teams, it was generally considered to have been a positive experience. Even the teams that reported having particularly struggled with learning it, or having been unwilling to initially devote resources towards doing so, felt that it had ultimately been useful:

*In retrospective, perhaps we would have had even greater progress with our project and higher learning outcome from the course if our understanding of SEMAT had improved at an earlier stage (Report 062)*

*When we later, a bit too late probably, actually sat down and studied what it meant and how to use it, it seemed kind of genius. (Report 044)*

## **4.2 Theme 2: Inexperience**

Another recurring theme present in the reports was inexperience in relation to SE. In their reports, the teams often discussed their own perceived inexperience with SE in relation to Essence. The inexperience of the teams evidently had a multifaceted significance to their experiences with Essence.

On one hand, the teams felt that Essence was *more* useful because they were inexperienced. They felt that, being inexperienced developers, Essence helped them (1) structure their way of working, (2) learn about new methods and practices, and (3) manage their projects better. In conjunction with the practice library, Essence was perceived to have been very educational in relation to SE methods and practices.

*While still being on our own and with little experience, SEMAT provided us guidelines that allowed us to improve and learn while planning and working on the project. Resulting in a much better experience with projects than before and a concept we are proud of. Knowledge we absolutely will include in future projects and programming. (Report 078)*

*...our experience with the ESSENCE kernel has been almost exclusively positive. Given that it prevents overlooking parts of the software development cycle, we perceived it as more beginner friendly than other competing, more fragmented approaches to software development methodology. (Report 047)*

On the other hand, some teams felt that their inexperience with SE might have also had a negative impact on the usefulness of Essence. As Essence encourages one to develop their own way of working, these teams felt they could not make the most of Essence due to their lack of knowledge about practices:

*A team of beginner developers such as ourselves might get locked up in the [practice] cards already made, resulting in using methods that is ineffective for us since we wouldn't make up any new techniques that isn't "available". We think that with a little more experienced team that hasn't made their own method yet, this would be extremely helpful. (Report 013)*

Not all teams considered this to be a negative situation, however. Some teams felt that the way Essence encouraged them to experiment with new practices and to learn by working as a team was helpful, even though they initially did not have a clear idea of what practices might work for their team. Essence, they felt, challenged them to actively think about what they were doing and why, and even though it did not provide direct answers to those questions, it facilitated learning in a positive manner. Thus, the general sentiment among the groups was that Essence, as well as the practice library related to it, had been very useful for them as inexperienced developers. As a concluding remark, it is worth noting that while not all of the teams comprised of individuals with little or no past experience with practical SE work, the resounding majority of them nonetheless did, being comprised of second year SE students. This was also evident in the way the teams actively reflected on their own inexperience in various ways in their reports.

### **4.3 Theme 3: Way of Working and the Method Prison**

One of the most discussed positive aspects of Essence perceived by the teams was its method-agnostic approach. The ability to freely choose between methods and practices

was considered both new and highly positive, letting them, in the words of Jacobson et al. (2017), escape the “method prison”:

*Our team really liked the freedom SEMAT gives you in defining the way you develop something and how you can customize it, choose the practices you want and not be forced to use practices you don't want to use (Report 036)*

*There were many positives of applying the kernel to our project, like choosing what we wanted to implement in our regular work day allowed us to use only what we wanted and thought we could benefit from. This level of freedom created a higher level of productivity than for example Scrum, where we are forced to use all aspects of the framework that do not necessarily benefit us. Not being forced to do things that we feel would slow us down and not benefit us really made us appreciate the SEMAT Essence Kernel (Report 071)*

As many of the students in the course had previously taken a course on Scrum, many of the reports consequently also included reflections related to Scrum. These teams discussed how they had initially started using Scrum or ScrumBut but had then begun to reflect on what they were doing and why, resulting in them refining their own way of working by using Essence. Used in conjunction with the practice card library, Essence provided them with new alternative practices to utilize. This resulted in the teams experimenting with different practices. On a more general level, they felt that the method-agnostic approach of Essence prepared them for different ways of working in the future.

Additionally, the teams reported positive experiences with actively reflecting on their way of working. Aside from initially tailoring a method for themselves, some of the teams reported having found Essence useful in facilitating the idea of continuously improving their work processes based on their experiences. Furthermore, some teams also noted that Essence had made it easier to communicate their way of working to the team as well as to discuss it within the team:

*This overview of all practices really benefited us when we put together our way of working and made it easy to visualize our workflow. Whenever a team member was unhappy with any aspect of our work methodology we reviewed the cards and added or removed any if needed. (Report 060)*

Finally, the teams discussed having learned much about new methods and practices simply by browsing through the practice cards available in the Ivar Jacobson practice library. This serves to underline the importance of tools related to adopting Essence. In this case, the practice cards helped teams of inexperienced developers tailor methods using Essence despite not having any previous experience with different SE practices.

#### **4.4 Theme 4: Progress Control**

The Essence kernel provides a framework upon which to build a project-specific tool. However, even without any modifications, the kernel already serves as a basic progress

control tool. This was also reflected in the reports. Most teams that had properly utilized the kernel had had positive experiences using Essence to manage and track progress:

*Selecting and using the alpha state cards that were relevant to our circumstances to assess our progress proved extremely effective. When we used them for the first time we were surprised to learn that we had not made as much progress as we thought. The cards were useful in seeing where we wanted to be in terms of progress in the different alphas, and thus facilitated the process of fixing our impediments. (Report 005)*

*The team then agreed to purchase a cork board and print out the Alpha State Cards in order to quickly and easily get an overview over the team's overall progress. This proved valuable, as none of the team members had partaken in any projects of this scale previously. The clear visualization the cards provided gave a much clearer picture of the project's progression overall than what the team found orally. (Report 055)*

Although Essence did clearly facilitate the idea of tailoring methods and choosing the methods that work best, this may not always be preferable. If the alternative to being locked in a “method prison” is the use of ineffective ad hoc methods, following an established method by the book may well be the more effective option. However, the teams felt that Essence helped them *formalize* their way of working aside from also facilitating the idea of tailoring it to suit their context-specific needs.

In relation to the inexperience of the teams discussed in a preceding sub-section, many of the teams felt that the Essence kernel provided a good overview of a software engineering endeavor *especially* because they had little experience with SE project work. Even though not all teams that utilized the kernel extended it, they nonetheless felt the Essence kernel in its base form was already useful in tracking their progress – except for one. One of the teams felt that they had had a solid understanding of the state of their project prior to using Essence and that “it didn’t help us anything to convert it into cards and more complicated sentences” (Report 059). This is not surprising as tools are just that: tools. Similarly, though formal methods and practices are typically preferred, it is quite possible to carry out SE endeavors using ad hoc methods, as e.g. a notable number of software startups chooses to do [17].

#### 4.5 Summary of Findings

Having discussed the results through the themes present in the data set, we now turn back to our formal research problem. Below, we provide summarizing answers for the two research questions posed in the introduction before going into more detail:

**RQ1:** Do bachelor level students find Essence useful?

**Results:** Essence was considered useful by the students, for varying reasons

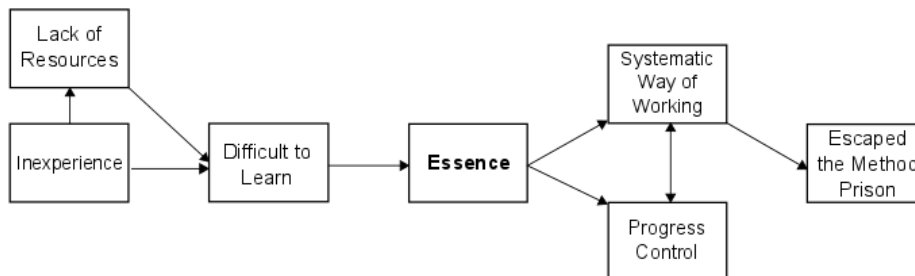
**RQ2:** What are the challenges in adopting Essence, specifically for inexperienced software developers, and what could be done to make its adoption easier?

**Results:** The largest challenge in adopting Essence was the lack of good tutorial resources, which consequently could be addressed by creating better such resources.

Though the student teams nearly universally considered Essence useful, there were differences between the teams in terms of *why* they considered it useful, largely based on the extent to which they had utilized it. Essence was considered useful for (1) teaching new methods and practices, (2) teaching a method-agnostic approach to SE, (3) helping the team properly structure their way of working, and (4) providing a useful framework for managing an SE project, depending on the degree of its utilization among each team. Few teams had anything negative to say about the specification itself, with most of the negative feedback relating to difficulties in adopting Essence.

Indeed, though Essence was considered useful by the teams, it was nonetheless evidently difficult for them to adopt. Many teams, even those that did utilize it the most, considered it to have been difficult to initially learn. The reports that discussed the reasons behind its perceived difficult adoption all cited the lack of good tutorial resources as the main problem. The teams felt that the resources they could find online were either hundreds of pages long or did simply not describe Essence simply enough for beginners. This resulted in some teams opting to focus their efforts elsewhere by e.g. focusing on learning to program and use programming tools, leaving Essence for later.

Having discussed our findings in relation to our research questions, we present a further, visual summary of how the themes discussed earlier in this section are inter-linked (Fig. 3). It is organized in a manner similar to how Giardino et al. [6] summarized their findings and depicts the adoption of Essence among students as a process. The student teams, as developers, were inexperienced. This inexperience resulted in a lack of resources as they had to divide their resources between e.g. learning to program, learning to use the programming tools, and learning Essence. In this situation, Essence often took on a lower priority, consequently becoming more difficult for the teams to learn. However, once the teams began to understand and utilize Essence, they began to work more systematically. All teams utilized Essence and the practice cards to work in a more systematic fashion, and many, but not all, teams grasped the kernel and began to use it as a progress control tool. For the teams that understood how to fully utilize Essence, its use ultimately resulted in an escape from the so-called method prison [10]. These teams actively reflected on their way of working and saw Essence also as a tool to facilitate learning in order to (attempt to) work in an efficient fashion in any given context in the future.



**Fig. 3.** Adoption process of Essence among SE students

Based on our findings, we therefore argue that SE students find Essence useful for multiple reasons. Furthermore, we confirm that Essence is considered difficult to learn, and our data suggests that the largest challenges in adopting Essence currently stem from a lack of tutorials and guides aimed at beginners. The current resources available online were considered too lengthy or advanced to be of use for new users of Essence.

## 5 Discussion

As extant literature has suggested [8], our findings confirm that Essence is indeed considered difficult and resource-intensive to adopt. However, our findings indicate that stems from a lack of good tutorial resources as opposed to Essence being difficult to use as such. The current manuals and other resources available were considered by the student teams to be too complex for beginners. Thus, the most direct solution to this issue would simply be the creation of better tutorial resources specifically aimed at new users of Essence.

As a solution to making Essence easier to adopt, [8] suggested the development of tools that could be used to make the practical use of Essence easier. This was not confirmed by our findings as none of the teams voiced explicit wishes for more tools to help utilize Essence. However, given that the practice card library, an external tool as well, was very positively received among the teams, it is likely that further tooling would also make Essence either easier to adopt and possibly more useful.

In terms of the usefulness of Essence for bachelor level students, our data indicates that Essence was indeed considered useful by the resounding majority of the project teams we studied. Less than ten teams out of 102 reported having found the use of Essence an outright negative and useless experience. In this light, we argue that Essence is useful for bachelor level students. More specifically, it was found useful in terms of (1) teaching new methods and practices, (2) teaching a method-agnostic approach to SE, (3) helping the team properly structure their way of working, and (4) providing a useful framework for managing an SE project.

From the point of view of SE education in universities, Essence is interesting as, based on our experiences, it can potentially provide a common ground for SE education through its method-agnostic nature. Such common ground is currently missing. We have showed that it can simultaneously teach students SE progress control as well as practical SE work. It also prepares SE students for working with different methods and practices out on the field. Essence could therefore be used to provide students with a higher-level understanding of the way SE work is structured. Essence can serve as a basis upon which SE students can build a general understanding of different SE methods as opposed to learning about single methods one at a time.

Learning to construct a method out of practices is an important learning goal for software engineering education. Based on our observations during the course, it was noted that some teams also learned to include so called anti-patterns or bad practices explicitly in their process description. This is a novel thought and should be further elaborated in future studies. By labeling a practice as a bad-practice, the team in question explicitly communicated about their improvement needs. Manual testing is an



example of such practice as it indicates lack of automated test suite, which slows down the development and is thus not a sustainable solution.

Additionally, in terms of generalizing our findings, we suggest that our findings could also be interesting for future research from the point of software startups. SE students, like startup practitioners [3, 12], are often more inexperienced developers, and it is also not uncommon for university students to participate in software startups during their studies. Most software startups fail [7] for various reasons, and Kon et al. [12] posited that specifically younger, more inexperienced startup practitioners are considered more prone to failure among investors. Software startups face various challenges across their life cycles [22], including challenges with “building product”, “staying focused & disciplined”, and “over capacity/too much to do”, which Essence could potentially be used to aid in solving. Finally, it has been established that software startups, like mature organizations, should concern themselves with structuring their work processes [19], which is something we found Essence to be useful for among SE students. Relating these past studies to our findings here, we suggest that future studies could investigate Essence from the point of view of software startups. Our findings, however, do not offer direct support to this link between these two contexts. In possibly pursuing this line of research, it could be useful to also evaluate the suitability of the Essence kernel in the context of software startups, as software startups have been shown to develop software in different ways than mature organizations [10], and their business aspect is linked with their SE process in a unique fashion.

Finally, while we have studied perceived difficulties in adopting Essence in the context of SE students, future studies may wish to study impediments to its adoption among practitioner organizations. As Essence has yet to see widespread practitioner adoption [21], the reasons behind this situation are worth investigating. Similarly, it is likely that more experienced practitioners find Essence useful or not useful for different reasons than the SE students studied in this paper.

## 5.1 Limitations of the Study

The primary limitations of the study are associated with the data collected during it. In collecting the data, we chose to rely on self-reported use of Essence over observation and regular check-ups. From this results that the validity of the reported utilization of Essence among the teams cannot be directly confirmed. However, the student teams seldom failed to report problems in utilizing Essence, with most teams that failed to utilize Essence fully reporting so themselves. In other cases, it was also largely possible to determine whether a team had understood the specification or not based on the way they reported on its utilization. We thus argue that this does not present a major threat to the validity of our data in such a large data set (102 teams).

Additionally, while the use of students as subjects for scientific studies is a long-standing topic of discussion across disciplines, including SE, the aim of this study was to study Essence specifically in relation to SE students and education. The use of students as subjects in this context is therefore not an issue.

## 6 Conclusions

In this paper, we have studied the Essence Theory of Software Engineering in a large-scale bachelor level course through experience reports. We introduced Essence to 102 project teams in a project-based SE course at a Norwegian university and observed its use during the projects. Based on 102 project reports discussing, among other things, the Essence use experiences of project teams of 4-5 individuals, we described the barriers of adoption of Essence and its usefulness for SE students.

We discovered that while Essence was considered difficult to learn by the teams, these difficulties largely stemmed from the lack of good tutorial resources. Some teams failed to fully utilize Essence, forgoing its progress control aspect partially or entirely, primarily due to its difficult adoption. There is thus a clear need for better introductory guides to Essence that are specifically designed for new users.

Past its difficult adoption, Essence was nonetheless nearly universally considered useful by the project teams. Even the teams that had not fully utilized Essence considered the method-agnostic approach and the practice cards to have been useful for planning out and formalizing their way of working during their projects. Additionally, the teams that had grasped the Essence kernel (except for two teams) also reported Essence having been useful in tracking progress during their projects. They felt that Essence gave them a good general understanding of SE project work through the alphas and that the alpha states helped them keep track of progress on their endeavor.

We therefore argue in favour of using Essence in SE education. By helping SE students gain a better understanding of SE project work and by preparing them for future adoption of various practices and methods, Essence can help tackle gaps [2, 13] between SE education and practice. To summarize our findings:

- (1) Essence can teach students new methods and practices by encouraging them to study them in order to tailor their own methods using Essence
- (2) Essence encourages students to adjust their way of working based on the SE context at hand as opposed to following existing methods by the book
- (3) Essence helps students structure their way of working in a practical setting
- (4) Better tutorial resources for Essence are needed to make it easier to adopt

## References

1. Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J.: Agile Software Development Methods: Review and Analysis. Otamedia: VTT Publications, 478 (2002).
2. Almi, N. E. A. M., Rahman, N. A., Purusothaman, D., and Sulaiman, S.: Software engineering education: The gap between industry's requirements and graduates' readiness. In: Computers & Informatics (ISCI), 2011 IEEE Symposium on (2011).
3. Crowne, M.: Why software startups fail and what to do about it – Evolution of software product development in startup companies. In: Proceedings International Engineering Management Conference (IEMC), 338-343 (2002).

4. Davis, F. D. Jr.: A Technology Acceptance Model for Empirically Testing New End-User Information Systems: Theory and Results. Massachusetts Institute of Technology (1985).
5. Ghanbari, H.: Investigating the causal mechanisms underlying the customization of software development methods. *Univ. of Jyväskylä: Jyväskylä Studies in Computing*, 258 (2017).
6. Giardino, C., Paternoster, N., Unterkalmsteiner, M., Gorschek, T., and Abrahamsson, P.: "Software Development in Startup Companies: The Greenfield Startup Model". *IEEE Transactions on Software Engineering*, 42(6), pp. 585-604 (2016).
7. Giardino, C., Wang, X., and Abrahamsson, P.: Why Early-Stage Software Startups Fail: A Behavioral Framework. In *International Conference of Software Business*, pp. 27-41. Springer, Cham (2014).
8. Graziotin, D., and Abrahamsson, P.: A Web-based modeling tool for the SEMAT Essence theory of Software Engineering. *Journal of Open Research Software*, 1 (2013).
9. Jacobson, I., Stimson, R., and Hastie, S.: Escaping Method Prison. <https://www.infoq.com/articles/escape-method-prison> last accessed 15 May 2018 (2017)
10. Jacobson, I., Ng, P., McMahon, P. E., Spence, I., and Lidman, S.: The Essence of Software Engineering: The SEMAT Kernel. *ACMQueue*, 10, pp. 40-52 (2012).
11. Kemell, K. O., Risku, J., Evensen, A., Dahl, A. M., Grytten, L., Jedryszek, A., Rostrup, P., Nguyen-Duc, A., and Abrahamsson, P.: Gamifying the Escape from the Engineering Method Prison - An Innovative Board Game to Teach the Essence Theory to Future Project Managers and Software Engineers. [to be published in the proceedings of ICE2018] (2018)
12. Kon, F., Cukier, D., Melo, C., Hazzan, O., and Yuklea, H.: A Panorama of the Israeli Software Startup Ecosystem. SSRN: <https://ssrn.com/abstract=2441157> (2014).
13. Lethbridge, T. C., Díaz-Herrera, J., LeBlanc Jr., R. J., and Thompson, J. B.: Improving software practice through education: Challenges and future trends. *Proceedings: FOSE '07 Future of Software Engineering* (2007).
14. Object Management Group: Essence – Kernel and Language for Software Engineering Methods. Version 1.1. <http://semat.org/essence-1.1>, last accessed 2018/05/28.
15. Park, J. S., McMahon, P. E., and Myburgh, B.: Scrum Powered by Essence. *ACM SIGSOFT Software Engineering Notes*, 41(1), pp. 1-8 (2016).
16. Parnin, C., Helms, E., Atlee, C., Boughton, H., Ghattas, M., Glover, A., Holman, J., Micco, J., Murphy, B., Savor, T., Stumm, M., Whitaker, S., and Williams, L.: The Top 10 Adages in Continuous Deployment. *IEEE Software*, 34(4), 86-95 (2017).
17. Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T. and Abrahamsson, P.: Software development in startup companies: A systematic mapping study. *Information and Software Technology*, 56, pp. 1200-1218 (2014).
18. Pieper, J.: Discovering the Essence of Software Engineering – An Integrated Game-Based Approach based on the SEMAT Essence Specification. In *Proceedings of the 2015 IEEE Global Engineering Education Conference (EDUCON)*, pp. 939–947 (2015).
19. Ries, E.: *The Lean Startups: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. New York: Crown Books (2011).
20. SEMAT: SEMAT and Essence – What is it and why should you care? <http://semat.org/what-is-it-and-why-should-you-care->, last accessed 2018/05/20.
21. SEMAT: Great pick up of Semat. [http://semat.org/news/-/asset\\_publisher/eaHEtyeuE9wP/content/great-pick-up-of-semat](http://semat.org/news/-/asset_publisher/eaHEtyeuE9wP/content/great-pick-up-of-semat), last accessed 2018/05/13.
22. Wang, X., Edison, H., Bajwa, S. S., Giardino, C., and Abrahamsson, P.: Key Challenges in Software Startups Across Life Cycle Stages. In: Sharp H., Hall T. (eds) *Agile Processes, in Software Engineering, and Extreme Programming. XP 2016. Lecture Notes in Business Information Processing*, vol 251. Springer, Cham (2016)



## II

# **SOFTWARE STARTUP PRACTICES: SOFTWARE DEVELOPMENT IN STARTUPS THROUGH THE LENS OF THE ESSENCE THEORY OF SOFTWARE ENGINEERING**

by

Kai-Kristian Kemell, Ville Ravaska, Anh Nguyen-Duc & Pekka Abrahamsson, 2020

*21st International Conference on Product-Focused Software Process Improvement (PROFES)* (pp. 402-418). Springer. Lecture Notes in Computer Science, 12562

DOI 10.1007/978-3-030-64148-1\_25

Reproduced with kind permission by Springer.

# Software Startup Practices – Software Development in Startups through the Lens of the Essence Theory of Software Engineering

Kai-Kristian Kemell<sup>1</sup>[0000-0002-0225-4560] Ville Ravaska<sup>1</sup>,  
Anh Nguyen-Duc<sup>2</sup>[0000-0002-7063-9200] and Pekka Abrahamsson<sup>1</sup>[0000-0002-4360-2226]

<sup>1</sup> University of Jyväskylä, Jyväskylä 40014, Finland

<sup>2</sup> University of Southeast Norway, Norway  
kai-kristian.o.kemell@jyu.fi

**Abstract.** Software startups continue to be important drivers of economy globally. As the initial investment required to found a new software company becomes smaller and smaller resulting from technological advances such as cloud technology, increasing numbers of new software startups are born. Startups are considered to differ from other types of software organizations in various ways, including software development. In this paper, we study software development in startups from the point of view of practices to better understand how startups develop software. Using extant literature and case study data, we devise a list of practices which we categorize using the Essence Theory of Software Engineering (Essence). Based on the data, we propose a list of common practices utilized by software startups. Additionally, we propose potential changes to Essence to make it better suited for the software startup context.

**Keywords:** Software Startup, Essence Theory of Software Engineering, Software Development, Software Development Practice, Case Study.

## 1 Introduction

Software startups continue to be important drivers of economy globally. As the initial investment required to found a new software company becomes smaller and smaller as a result of technological progress, more and more startups are founded. While most startups fail [4], just like most new companies [13], some go on to become mature, established software organizations, or even multinational technology giants.

Typically, the main argument for studying software startups is that they differ from mature software organizations in various ways, thus making the findings of many existing studies not directly applicable to them. This is a result of there still being no accurate definition for what a startup is [21][23]. Various characteristics such as time pressure or resource scarcity are attributed to startups to differentiate them from mature companies [21], but academically drawing an exact line has been a challenge in the area [13]. The way software startups develop software has been one area of study.

For example, Paternoster et al. [21] conducted a more general, large-scale study aiming to understand how software startups develop software. They noted that software startups operate mostly using various Agile practices or ad hoc methods. Specific facets of software development (SWD) in software startups, such as prototyping [19] have also been studied. However, studies focusing on Software Engineering (SE) practices in software startups are still scarce, and studies into SWD in software startups in general are still needed [23]. Some high-profile startup practices such as the Five Whys are commonly discussed in e.g. startup education, but systematic studies are lacking.

Thus, to better understand how software startups develop software, we study practices in this paper. Specifically, we seek to understand what practices are commonly used by software startups. In addition, we approach this topic through the lens of the Essence Theory of Software Engineering and seek to understand how this theory fits into the context of software startups. To this end, we study how the seven alphas of the theory (section 2.3) fit the context of software startups, and whether other alphas would be needed to make the theory better suited for this context.

## **2 Background – Software Startups, Software Development Practices, and the Essence Theory of Software Engineering**

This section is split into three subsections. First, we discuss SWD in software startups. Then, we define SWD practices in this context. Finally, we discuss Essence.

### **2.1 Software Development in Software Startups**

Typically, software startups do not strictly follow any formal software development method [21]. Instead, they combine practices from different methods that suit their needs at the moment or simply use ad hoc practices [18].

As the aim of this study is to uncover software development practices universal to (most) software startups, a notable paper is that of Dande et al. [6]. Dande et al. [6] studied software startups in Finland and Switzerland and devised a list of 63 practices commonly utilized by software startups. However, these practices are not solely software development ones but also include practices related to customers and business. Kamulegeya et al. [11] studied these practices and reported that they seemed to apply in the Ugandan startup context as well, further validating this list of practices. They do add, however, that culture and location might influence commonly used practices.

Other studies focusing on practices have not aimed to create such extensive lists of practices but have nonetheless studied software startup practices in different contexts. Klotins, Unterkalmsteiner, and Gorschek [15], for example, created a framework for categorizing software startup practices that differs from the one proposed by Dande et al. [6]. Giardino et al. [9] propose the Greenfield Startup Model to explain software development in early-stage software startups. In the process, they uncovered various practices that supplement and confirm the findings of Dande et al. [6]. Paternoster et al. [21] in their study on how software startups develop software discuss having found 213

practices, which, however, were not listed in their paper. Nonetheless, their findings to lend support to those of Dande et al. [6].

## 2.2 Software Development Practice as a Construct

Jacobson et al. [10] suggest that a set of practices is what forms a method in the context of SE. Methods, according to them, describe ways-of-working, i.e. how work should be carried out. A way-of-working exists in an organization even if a formal SE method is not utilized [10]. A practice, then, describes a more atomic unit of work.

Historically in academic literature, and particularly in Information Systems, the construct *technique* has been used for the same purpose in the context of method engineering [22]. Tolvanen [22] defines a technique to be a set of steps and rules that define how a representation of information system is derived and handled using conceptual structure and related notation. A tool, in this context, refers to a computer-based application supporting the use of a technique.

## 2.3 The Essence Theory of Software Engineering

The Essence Theory of Software Engineering [10] provides a way of describing methods and practices. It consists of a notational language and a so-called kernel, which includes building blocks that can be used as a basis for constructing methods. The kernel, its authors argue [10], contains basic elements that are universal in any SE project.

The Essence kernel contains three types of objects: alphas (i.e. things to work with), activities (i.e. things to do), and competencies (skills required to carry out the work). In this study, we focus on the alphas in the context of software startups. The seven Essence alphas are as follows: (1) Stakeholders, (2) Opportunity, (3) Requirements, (4) Software System, (5) Team, (6) Way of Working, and (7) Work. These alphas are split into three areas of concern. The first two belong in the customer area of concern, numbers three and four in the solution area of concern, and the last three in the endeavor area of concern. Furthermore, each alpha has alpha states used to track progress on the alpha. [10]

The authors of Essence posit [10] that these are the essential elements that are present in every SE project. Every project, then, has its own unique context, which most likely contains more things to work with, but those are not universal to every project. In order to reap the most benefits out of Essence, its users would then extend this basic kernel with the Essence language to include these unique features of their particular project or company to describe their method(s) with it.

In this paper, the role of Essence is two-fold. First, it serves as a framework for analyzing our data. We utilize the alphas to sort the software startup practices we discover into categories. Secondly, in the process of doing so, we study whether all the uncovered practices fit into these seven alphas. I.e., do the alphas also present all the essential elements of software development in software startups?

We chose to utilize Essence as the framework for this study for two reasons. First, Essence is an OMG standard. Standards can shape the industry and should be studied. In this case, we are particularly interested in seeing whether Essence suits startups as well. Secondly, Essence provides one framework for categorizing work in SE projects

through its kernel and alphas. In studying practices, we considered it important that we have a framework for categorizing them in some fashion.

### 3 Study Design

The goal of this study is outlined at the end of the introduction. We approached this topic using a qualitative multiple case study approach. Aside from this empirical data, we utilized the list of 63 startup practices presented by Dande et al. [6].

#### 3.1 Data Collection

The empirical data for this study was collected by means of a multiple case study (n=13) (Table 1). The interviews were conducted F2F. The audio was recorded, and the recordings were transcribed for analysis. All the respondents were CEOs or founders, as we wished to interview respondents with extensive knowledge of the case startups.

**Table 1.** Cases.

Case	Employees	Company Domain	Respondents	Age (in years, at the time of interview)
1	6	Software/ Hardware	1	<1
2	5	Software	3	1-3
3	3	Software / Hardware	2	<1
4	5	Software	1	1-3
5	7	Software / Consulting	1	<1
6	3	Software / Hardware	1	1-3
7	8	Software	1	>3
8	12	Software	1	>3
9	6	Software	1	1-3
10	5	Software	1	>3
11	85	Software / Hardware	1	1-3
12	5	Software / Hardware	1	>3
13	6	Software	1	>3

We utilized a qualitative, thematic interview approach. We chose a thematic approach because most software startups develop software ad hoc [18][21]. Data were then collected with one of two interview instruments depending on how technical the respondent(s) were. With technical respondents, we utilized an interview instrument (found on Figshare<sup>1</sup>) more focused on the technical aspects of software development (interviews 6 to 13 in Table 1). With less technical respondents and in group interviews, we utilized an interview instrument built around the Essence alphas (same Figshare link below).

In utilizing two interview instruments, we wanted to gain a deeper understanding of the practices used through triangulation in terms of data collection methods, as suggested by Langley [16] in the context of process data. Using different types of data can

<sup>1</sup> <https://doi.org/10.6084/m9.figshare.13017227.v1>



provide a more comprehensive understanding of the phenomenon. In this case, we felt that focusing solely on the technical aspects might omit some less technical practices.

### 3.2 Data Analysis

The analysis of the empirical material in this paper was conducted following the thematic synthesis guidelines of Cruzes and Dyba [5]. The material was first transcribed for analysis. The material was then read thoroughly for an initial overview of the data. After this, the coding process was started, and each interview was coded. These codes were then arranged into themes. The coding process was done inductively, with codes and themes arising from the data (as opposed to e.g. using Essence as the framework at this stage). E.g., codes included such codes ‘team’, ‘funding’, and ‘prototype’. Using this approach, we analyzed the data to find practices, either ones already discussed by Dande et al. [6] or novel ones, with the novel ones made into a list.

Practices that were discussed by two or more of the case startups were considered prevalent enough to be included into the list of practices. Once the empirical data had been analyzed and new practices had been formulated, we took the list of 63 software startup practices of Dande et al. [6] and these new practices and inserted them into the framework of the Essence Theory of Software Engineering [10] and its alphas. I.e., we categorized each practice, if possible, under one of these alphas (see section 5.2 for critical discussion about this approach). The categorized practices were then reviewed by three other authors to form a consensus.

## 4 Results

This section is divided into 9 subsections. In the first one, we present the new practices we uncovered through the case study. In the next seven, we go over the results in relation to each Essence alpha, discussing the practices found in each category. In the ninth and final one, we discuss practices that did not fit under any of these alphas.

Given the space limitations of this paper, the clarifying descriptions for the 63 practices of Dande et al. [6] have not been included in the tables in this section. Such descriptions have, on the other hand, been added for any novel practices proposed by us. Each practice has an identified (Pn), where practices P64 and up are practices based on the empirical data and practices P63 and below are from Dande et al. [6].

### 4.1 New Practices

Based on the data, we propose 13 new practices (Table 2) that were not present in the list of Dande et al. [6]. These practices were mentioned by at least two case startups. Other new practices were also uncovered but discussed by only one case startup. These practices were not considered common based on this set of data.

**Table 2.** New practices based on our data.

<b>ID</b>	<b>Practice</b>	<b>Description</b>
P64	Study subjects that support the startup	Studying while working on a startup gains competence in the team without growing in personnel.
P65	Attend startup events	Startup events provide opportunity for feedback from experts and allows you to meet potential investors.
P66	Create an MVP early on	MVP helps you to focus on the most important features in the beginning.
P67	Test features with customers	Testing features with real customers gets you the best feedback.
P68	Get advisors	Experienced professionals or investors can help startup to grow in advisor or mentor role.
P69	Use efficient tools to plan your business model	Business model canvas, pitch deck etc. help you to focus your business idea and are easy to change if needed.
P70	Test different tools	Start with tools team is familiar with and test different ones to find those that work the best for you.
P71	Conduct market research	Research the markets and competitors to focus your idea and to find your unique value proposition.
P72	Have frequent meetings with the whole team	Use meetings to organize and plan your work at least once a week.
P73	Avoid strict roles	Let the team co-operate in all of the tasks.
P74	Create a prototype	Create prototype to validate your product or features.
P75	Use efficient communication tools	Use tools that allow natural communication inside the team when not working in the same space.
P76	Prioritize features	Choose which features are needed now and plan others for future releases.

## 4.2 Opportunity

The opportunity alpha is related to understanding the needs the system is to fulfill and is within the customer area of concern. Practices for this alpha are presented in Table 3 below. No new practices for this category were found in the data.

**Table 3.** Practices for the Opportunity alpha.

<b>ID</b>	<b>Practice</b>	<b>Cases Supporting</b>
P1	Focus your product	1,2,6,7,8,9,11,12,13
P2	Find your value proposition and stick to it on all levels	9,13
P4	Focus on goals, whys	9
P18	Validate that your product sells	1,2,4,5,7,8,11
P20	Form deep relations with the first customers to really understand their needs	1,6,9,11,13
P33	In the development of customer solutions, find a unique value proposition in your way of acting	1,2,3,5,6,8,9
P34	Follow communities	1,2

The case startups were highly focused on understanding their customers and fulfilling the needs of the customer (segments). This is in line with the idea of software startups

being product-oriented and customer-focused. On the other hand, the lack of support for P4 makes it seem that these startups were more focused on fulfilling the needs they had uncovered rather than understanding why these needs were important.

Focusing on the system and the needs it was intended to fulfill was considered important from the point of view of competition as well. Focusing on one's unique value proposition is conventionally considered an important strategy for differentiating from one's competitors.

### 4.3 Stakeholders

Four practices were categorized under the stakeholder alpha (Table 4), which is another alpha in the customer area of concern in Essence. For startups, most notable stakeholders are typically investors and customers or users. In addition, nearly half of the case startups discussed the importance of their advisors as stakeholders (P68).

**Table 4.** Practices for the Stakeholders alpha.

<b>ID</b>	<b>Practice</b>	<b>Cases Supporting</b>
P24	Keep customer communications simple and natural	6
P32	Showing alternatives is the highest proof of expertise	-
P35	Share ideas and get more back	1,2
P68	Get advisors	1,4,5,6,8,9

Especially early-stage startups tend to rely on advisors. For example, startup ecosystems tend to foster advisor relationships in various ways. Startups working in incubators are likely to receive guidance from various experts. Advisors can provide startups with capabilities they are lacking and help them expand their contact networks.

The practice of sharing ideas to hone them and to get feedback was also discussed by some case startups. While in some cases companies may be reluctant to share their ideas in fears of having them stolen, none of the case startups indicated this type of concerns. To this end, advisors can also provide feedback if a startup is afraid of revealing their ideas to potential investors due to such concerns.

### 4.4 Requirements

Requirements help provide scope for the work being done on the system. Four new practices were uncovered in this category and most existing practices in this category were well-supported by the cases (Table 5).

However, P3 was in conflict of what some of the case startups stated. P3 posits that a startup should present its product as facilitating rather than competing. While this is one valid approach, startups do also seek to compete in some cases.

The requirements alpha, in the data, was closely related to the stakeholders alpha: uncovering customer needs was the main focus in requirements (P10). In the case startups, prototypes were typically used to do carry out validation (P67, P74). While a startup should be open to new features and needs (P51), they should be prioritized (P76) to create a clear core product (P52, P53).

**Table 5.** Practices for the Requirements alpha.

<b>ID</b>	<b>Practice</b>	<b>Cases supporting</b>	<b>Cases conflicting</b>
P3	Present the product as facilitating rather than competing to the competitors	-	1,2,6
P5	Use proven UX methods	12	-
P10	Design and conduct experiments to find out about user preferences	1,2,4,6,9,12,13	-
P21	Use planning tools that really show value provided to customers	2	-
P51	Anything goes in product planning	1,2,11	-
P52	To minimize problems with changes and variations develop a very focused concept	1,2,3,4,5,6,7,12,13	-
P53	Develop only what is needed now	1,2,3,12	-
P66	Create an MVP in the beginning	1,2,4,13	-
P67	Test features with customers	1,3,4,5,6,7,8,9,11	-
P74	Create prototype	1,2,3,4,5,6,9,12	-
P76	Prioritize features	1,2,3,9,11	-

#### 4.5 Software System

The software system alpha is focused on the product itself, i.e. the system; software or hardware. The software system alpha is in the solution area of concern of the Essence kernel. Some of the previously proposed practices were largely prevalent in the cases while some received little support from our data. More technical practices (P23, P54, P57) would have required a more technical focus from the interviews. No new practices were proposed for this category. The practices for this category are in Table 6.

**Table 6.** Practices for the Software System alpha.

<b>ID</b>	<b>Practice</b>	<b>Cases supporting</b>	<b>Cases conflicting</b>
P7	Have a single product, no per customer variants	1,2,3,5,7,8,11,12	6,13
P8	Restrict the number of platforms that your product works on	1,2,3,4,7,12	-
P14	Anyone can release and stop release	2	-
P23	Adapt your release cycles to the culture of your users	-	-
P54	Make features easy to remove	-	-
P55	Use extendable product architecture	1,2,3,9,11	-
P57	Bughunt	-	-
P58	Test APIs automatically, UIs manually	2,13	-
P59	Use generic, non-proprietary technologies	2,7	-
P60	Create a solid platform	3,8,9,11	-

Out of the practices of this category, only P7 had some conflicts in the data. This practice is largely B2C focused, whereas a B2B startup might understandably focus on tailoring its system especially for larger customers. However, it is perhaps worth aiming for a modular product where such manual tailoring is not needed.

Overall, these practices further underlined that startups should have a clear focus in their development. For example, they should focus on a limited number of platforms, possibly only one initially (P8). Additionally, startups are conventionally seen as agile and their systems as prone to changes based on feedback. Indeed, these practices support the idea that the system should be developed with modifications in mind (P60). Features should be easily added (P55) or removed (P54) when necessary.

#### 4.6 Work

Work in the context of Essence refers to the work tasks required to produce the system. It is under the endeavor area of concern in the Essence kernel. For software startups, this also involves business model development. How the work is carried out from the point of view of e.g. methods, belongs into the way of working category, on the other hand. Few existing practices were considered to belong into this category and no new practices for this category were found (Table 7).

**Table 7.** Practices for the Work alpha.

<b>ID</b>	<b>Practice</b>	<b>Cases supporting</b>
P44	Tailored gates and done criteria	8
P48	Fail fast, stop and fix	1
P62	Use the most efficient programming languages and platforms	2,3,7

While P48 is arguably closely related to prototyping and validation activities which were extensively discussed by the respondents, it was seldom discussed directly. On the other hand, P62 was discussed in relation to system architecture. Efficiency in this case was considered subjectively: the developers focused on languages and platforms they had prior experience with and could thus start working the fastest with.

#### 4.7 Team

The team comprises the individuals working on the startup, the founders or owners and the employees or unpaid ones. It is under the endeavor area of concern in the Essence kernel. The team sizes for the case startups are in Table 1 in Section 3. One new practice (P64) was added into this category based on the data (Table 8).

The most mentioned practices were P41 and P42. The initial team is important as it needs to have the required competencies (P41). To this end, an experienced team may be required (P42). Some of the cases conflicted with P42, although not because the teams did not want an experienced team but simply because they could not find one.

However, this did not mean that the startups did not want an experienced team. Rather, they simply did not have one due to being founded by a group of students with little prior experience.

If the team is lacking competencies and expanding the team is not possible or feasible in a given situation, the existing team members may have to learn new skills instead (P64). This also ties to P37, as the small team sizes often result in a single

employee having to take on various different tasks. A developer is often involved in business decisions as well, especially in early-stage startups.

Flat organization structures (P26) are associated with startups and this was also the case in our data. Involving employees in decision-making may also serve to better bind them (P29). With a small, focused team, staff turnover can be damaging (P38).

**Table 8.** Practices for the Team alpha.

ID	Practice	Cases supporting	Cases conflicting
P26	Flat organization	1,2,3,5,9	-
P27	Consider career expectations of good people	4,9	-
P28	Don't grow in personnel	1,2,3,12	-
P29	Bind key people	2,3,6,7	-
P36	Small co-located teams	1,2,3,4,5,6	12
P37	Have multi-skilled developers	1,2,3,12	-
P38	Keep teams stable in growth mode	1,2,3,4,6,7,13	9
P40	Sharing competence in team	4,5	-
P41	Start with competence focus and expand as needed	1,2,3,4,6,8,9,13	-
P42	Start with small experienced team and expand as needed	1,2,3,4,7,8,12,13	1,2,3
P64	Study skills and topics that support your startup	1,2,3,4,8,9	-

#### 4.8 Way of Working

Way of Working refers to how the work is carried out, including practices, tools, processes, and methods [10]. It is under the endeavor area of concern in the Essence kernel. Most previously proposed practices were supported by our data in this category. Four new practices were proposed for this category (Table 9).

Most case startups discussed having taken some existing agile practices and tailoring them rather than using them by the book (P47). While this ties to P72 in that frequent team meetings are common in agile development, it gained enough emphasis to be its own separate practice. On the other hand, the use of by-the-book methods (P46) was not discussed by any of the startups, with the startups using various mixed practices. Communication in general is an important part of agile development, and arguably development in general. The case startups frequently discussed the importance of tools in facilitating communication (P75). While shared physical workspaces can reduce the need for tools, their importance is highlighted when working remotely. An early-stage startup may not have a physical workspace at all, or its members may have erratic work hours due to having a day job, resulting in communication tools becoming important.

Self-organizing teams are recommended in agile development and this is also arguably common for startups (P39, P73).

**Table 9.** Practices for the Way of Working alpha.

ID	Practice	Cases supporting	Cases conflicting
P9	Use enabling specifications	1,2,3	-
P15	Create the development culture before processes	1,8,11	-
P39	Let teams self-select	1,2,3,5,8	-
P43	Have different processes for different goals	-	-
P45	Time process improvements right	3	-
P46	Find the overall development approach that fits your company and its business	-	-
P47	Tailor common agile practices for your culture and needs	1,2,3,4,6,7,8,13	-
P49	Move fast and break things	4,7	-
P50	Forget Software Engineering	1	-
P61	Choose scalable technologies	2,3,9,11	-
P63	Start with familiar technologies and processes	1,2,3,7	-
P70	Test different tools	1,3	-
P72	Have frequent meetings with the whole team	1,2,3,4,5,8,12	-
P73	Don't have strict roles	1,2,3	9
P75	Use efficient communication tools	2,3,5	-

#### 4.9 Other Practices Unsuitable for Existing Essence Alphas

Not all of the practices we propose, or the ones proposed by Dande et al. [6], fit under any of the existing Essence alphas. These were practices related to the business aspect of software startups, such as marketing, business model development, or funding. Whereas Essence focuses on SE in mature software organizations, the business aspect in software startups is closely intertwined with software development. For example, the needs of the customers or the customers in general, may not be clear to a software startup, which results in the requirements evolving over time.

Practices P6, P11, P25, P31, and P71 concern marketing activities. For example, P25 is about getting a few initial customers who are particularly interested in the system and who can then be used as reference customers in marketing, or who themselves can market the product. P6 and P31 are more general marketing practices. These types of activities are difficult to incorporate into any existing Essence alpha. While marketing is a customer related activity and thus could be linked to stakeholders, the existing stakeholder alpha focuses on clearly identified and involved stakeholders such as the organization commissioning a project, as opposed to obtaining new customers (stakeholders).

P16 and P17 are related to funding. Funding or simply available cash to burn is something that is constantly tracked in a startup, much like the alphas are tracked in Essence. No existing alpha supports funding with clear emphasis. Some of the alpha states of the Work alpha include mentions of securing sufficient funding, but this process is seldom so straightforward in a startup.

The remaining practices in this category are related to overall business model development and business planning. For example, P13 suggests that outsourcing some part of the business can help the startup focus on the core product, and P22 suggests a strat-

egy for rapid and high growth. P30, on the other hand, could be filed under the Stakeholders alpha, but doing so might not place sufficient emphasis on the strategic importance of such decisions from a business point of view.

As we do not formally develop new alphas in this paper, we leave the proposals related to these observations for the following discussion section.

**Table 10.** Practices not applicable to any existing Essence alpha.

ID	Practice	Case supporting	Case conflicting
P6	Do something spectacular	-	-
P11	Use tools to collect data about user behavior	1,2,7	-
P12	Make your idea into a product	1,2,3,4,5,6,7,8,12,13	11
P13	Outsource your growth	5,9,11,12,13	3
P16	Get venture capital and push your product	1,2,4,5,8,9	3
P17	Fund it yourself	1,2,3,7,9	-
P19	Focus early on those people who will give you income in the long run	5,6,7,8,11,13	-
P22	Start locally grow globally	1,2,3,6,7,8,9,13	-
P25	Help customers create a great showcase for you with support	1,6,8,9	-
P30	Form partnerships and bonds with other startups	1,3,4,5,13	-
P31	Make your own strength as a “brand”	8	-
P56	Only use reliable metrics	5,6,7	-
P65	Attend startup events	1,2,3,4,8	-
P69	Use efficient tools to plan your business model	1,2,3	-
P71	Conduct market research	1,2,6,12	-

## 5 Discussion

The primary contributions of this study are (1) this list of practices 76 and its implications we discuss here, and (2) the implications these practices have for utilizing Essence in the startup context. First, In terms of the practices and the data overall, our findings seem to support existing literature. Paternoster et al. [21] argued that startups develop software using various agile practices or ad hoc. The case startups of this study did discuss the utilization of methods either, only occasionally mentioning singular practices that could be seen as Agile. Many of the practices, such as focusing on a set of functionalities or utilizing MVPs, are also discussed in the Greenfield Startup Model of Giardino et al. [9].

It is common for larger software organizations, too, to take a method such as SCRUM and then omit some practices to create yet another “scrumbut,” with quality practices often the first ones to go [8]. Startups, on the other hand, seem to seldom even use tailored methods, pointing to an even higher degree of unsystematic approaches to SE – based on both our data and existing studies (e.g. [18][21]).

In terms of how startups differ from mature organizations, aside from the aforementioned use of ad hoc methods and singular agile practices, technical debt is one element



typically associated with startups [1][9]. Some of the practices were ones that would arguably generate technical debt (e.g. "move fast, break things"), but the case startups did not explicitly discuss technical debt as an issue.

The list of practices in this paper presents a closer look at the way software startups develop software. These existing studies have focused on method use and specific issues faced by startups such as technical debt accumulation, or MVPs. By better understanding what practices startups use we can further our understanding of how they differ from larger software organizations. This is arguably important as it possible that one factor contributing to the lack of method use in startups may be that they feel that existing methods are not well-suited for the startup context. The practices listed in this paper support existing literature. For example, P66 posits that an MVP should built early on, which is in line with Klotins et al. [14] who argue that one common issue for software startups is taking too long with an initial version of the product.

The other contribution of this paper is related to Essence, which we have used as a theoretical framework for categorizing the practices in this paper. Essence is intended to be used in any SE endeavor. Its so-called kernel, its authors argue [10], contains the elements present in every SE endeavor. This kernel acts as a set of building blocks that can then be extended using the Essence language to describe methods.

In this paper, we looked at Essence from the point of view of software startups. Based on our data and extant literature (e.g. [14, 15]), the business aspect is deeply intertwined with software development in the startup context. In fact, Klotins et al. [14] argue that software startups largely fail due to business issues that originate from SE processes. This supports the idea that SE and business aspects are difficult to separate in software startups. If the goal of Essence is to contain the elements present in every SE endeavor, for the startup context this would thus seem to include business elements.

For example, a conventional software project that is commissioned has clear requirements which have been agreed upon with the customer(s). On the other hand, software startups spend significant effort trying to ascertain whether their idea addresses a real need of a real customer (segment) at all. These idea or business validation activities to hand-in-hand with development activities. Moreover, whereas a developer in a large organization simply develops, in startups roles are seldom so clear-cut, especially early on. In an early-stage startup, a developer may be involved in business activities as well.

Some of the practices in this paper, namely the business-related ones, were not well-suited for any existing Essence alpha. To better incorporate the business aspect into Essence in order to make it more suitable for the startup context, we propose the following: (1) a fourth area of concern for business aspects should be added, and (2) new alphas for this business area of concern should be added. We suggest that funding, business model, and marketing could be new alphas for this area of concern.

Alphas are things to work with and while using Essence one tracks progress on the alphas, each of which is split into alpha states to aid in this process. Therefore, each of these three new alphas should be in some way measurable. First, funding pivotal for any startup [3], and can be quantitatively measured with various metrics, making it a straightforward alpha. Progress on this alpha is likely to fluctuate as cash is burned and new funding is obtained. Secondly, business model development is at the core of a startup [17]. Indeed, one widely used definition for what is a startup posits that a startup

is a “temporary organization designed to look for a business model that is repeatable and scalable” [2]. Startups constantly invest resources into validating that they are trying to address a real need. Progress on business model development could be tracked by evaluating how well the current business model is functioning and to what extent it is already operational. Thirdly and finally, marketing may warrant its own alpha. Marketing is as important to startups as it is to any other company [4]. Startups generally have less capital to spend on marketing, forcing them to get creative with it.

Alternatively, one other option would be to look at other theories and frameworks commonly utilized by startups for business model development. Potential business-related alphas could be derived e.g. from the Business Model Canvas [20].

### 5.1 Practical Implications

The primary practical contribution of this study are the practices listed in the tables in the results section. These practices can help guide work in software startups. Moreover, they can be used to construct methods in conjunction with other practices. Additionally, based on these practices and the data, we suggest the following implications:

- Flat organization and self-organizing teams seem to be an effective way for constructing the initial team. Self-organizing teams have been noted to be beneficial in Agile [12]. It may also be beneficial to avoid strict roles.
- You should have a clear idea of what is the core product and what features are the key features at any given moment. Having a scope too large for the product or an MVP is a frequent reason for failure in software startups [14].
- Forming close relationships with initial customers and users is beneficial. They can help you develop your product and participate in development. They can also aid in marketing. For example, user communities on social media platforms built around your (future) product can be beneficial in various ways.

### 5.2 Limitations of the Study

There are several limitations in this study. First, defining practices is a challenge in various ways. The level of abstraction in defining a practice can be subjective, and a single practice, when trying to describe how work should be carried out, can be described with varying levels of detail. Thus, some practices could be combined under a single practice of a higher level of abstraction rather than being split into multiple, more detailed practices. This is something that should be taken into account when looking at the practices discussed in this paper.

Secondly, practices in Essence can belong under multiple alphas. For the clarity of presentation, we chose to separate them into categories by alpha. However, some practices under one alpha could also justifiably be assigned under another alpha. Thus, the categorization in this paper is not conclusive and was used to 1) structure the analysis section, and 2) to evaluate whether each practice would fit under *any* existing alpha. Some of the business-focused practices could not clearly fit under any existing alpha, which was one of the main contributions of this study.

Thirdly, eliciting practices is also a challenge. Aside from practices explicitly considered practices by the respondents (e.g. pair programming), practices need to be defined based on what the respondents tell about their startup and its team and their work. This, too, is not a fully process if the practices are defined by an external party (researchers). We present but one way of categorizing work in startups into practices. Indeed, though they never listed them, Paternoster et al. [21] report to have found 213 practices, indicating that many more practices could be outlined based on different data.

Finally, qualitative studies can suffer from generalizability issues due to the nature of the approach. We, however, argue that 13 cases is a large enough number for some generalizability. E.g., Eisenhardt [7] suggests five cases to be sufficient for novel areas.

## 6 Conclusions

In this paper, we have studied Software Engineering (SE) in software startups from the point of view of practices, by means of a case study of 13 startups. Data were collected through semi-structured interviews. This set of data was used to complement and expand upon the results of an existing study that produced a list of 63 practices [6]. Based on our empirical data and this list, we propose 76 software startup practices that can be used in method engineering in the startup context.

We then took these practices and inserted them into the framework of the Essence Theory of Software Engineering to understand whether Essence also covers the aspects of SE in software startups and not just conventional SE projects. Our results suggest that the business aspect of startups is so intertwined with SE that the more business-oriented practices could not fit into the framework of Essence. We propose that Essence either be extended to include these business aspects for the startup context, or that other theories and tools are used in conjunction with it to cover the business aspect. We propose potential new alphas that could be used to extend Essence that future studies.

## References

1. Besker, T., Martini, A., Lokuge, R. E., Blincoe, K., Bosch, J.: Embracing Technical Debt, from a Startup Company Perspective. In Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE (2018).
2. Blank, S.: The four steps to the epiphany: successful strategies for products that win. Book-Baby (2007).
3. Chang, S. J.: Venture capital financing, strategic alliances, and the initial public offerings of Internet startups. *Journal of Business Venturing*, 19(5), 721-741, (2004).
4. Crowne, M.: Why software product startups fail and what to do about it. Evolution of software product development in startup companies. In Proceedings of the 2002 Engineering Management Conference IEMC'02, pp. 338-343, IEEE (2002).
5. Cruzes, D. S., Dyba, T.: Recommended steps for thematic synthesis in software engineering. In Proceedings of the 2011 Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 275-284, IEEE (2011).

6. Dande, A., Eloranta, V. P., Kovalainen, A. J., Lehtonen, T., Leppänen, M., Salmimaa, T., ... Koskimies, K.: Software startup patterns - an empirical study. Tampereen teknillinen yliopisto. Tietotekniikan laitos. Raportti-Tampere University of Technology. Department of Pervasive Computing. Report; 4 (2014).
7. Eisenhardt, K. M.: Building theories from case study research. *Academy of Management Review*, 14(4), 532-550 (1989).
8. Ghanbari, H., Vartiainen, T., Siponen, M.: Omission of Quality Software Development Practices: A Systematic Literature Review. *ACM Computing Surveys*, 51(2), (2018).
9. Giardino, C., Paternoster, N., Unterkalmsteiner, M., Gorschek, T., Abrahamsson, P.: Software Development in Startup Companies: The Greenfield Startup Model. *IEEE Transactions on Software Engineering*, 42(6), 585-604 (2016).
10. Jacobson, I., Ng, P. W., McMahon, P., Spence, I., Lidman, S.: The essence of software engineering: the SEMAT kernel. *ACM Queue*, 10(10), 40 (2012).
11. Kamulegeya, G., Hebig, R., Hammouda, I., Chaudron, M., Mugwanya, R.: Exploring the Applicability of Software Startup Patterns in the Ugandan Context. In *Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 116-124, IEEE, (2017).
12. Karhatsu, H., Ikonen, M., Kettunen, P., Fagerholm, F., Abrahamsson, P.: Building Blocks for Self-Organizing Software Development Teams a Framework Model and Empirical Pilot Study. In *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE)* (2010).
13. Klotins, E.: Software start-ups through an empirical lens: are start-ups snowflakes? In *Proceedings of the International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms (SiBW)* (2018).
14. Klotins, E., Unterkalmsteiner, M., Gorschek, T.: Software Engineering Antipatterns in start-ups. *IEEE Software*, 36(2), 118-126, (2018).
15. Klotins, E., Unterkalmsteiner, M., Gorschek, T.: Software engineering in start-up companies: An analysis of 88 experience reports. *Empirical Software Engineering*, 24(1), 68-102. (2019)
16. Langley, A.: Strategies for Theorizing from Process Data. *Academy of Management Review*, 24(4), (1999).
17. Lueg, R., Malinauskaite, L., Marinova, I.: The vital role of business processes for a business model: the case of a startup company. *Problems and Perspectives in Management*, (12, Iss. 4 (contin.)), 213-220, (2014).
18. Melegati, J., Goldman, A., Paulo, S.: Requirements Engineering in Software Startups: a Grounded Theory approach. 2nd Int. Work. Softw. Startups, Trondheim, Norw. (2016).
19. Nguyen-Duc, A., Wang, X., Abrahamsson, P.: What Influences the Speed of Prototyping? An Empirical Investigation of Twenty Software Startups. In *Proceedings of the 2017 International Conference on Agile Software Development (XP2017)*, pp. 20-36. (2017).
20. Osterwalder, A., Pigneur, Y., Clark, T.: *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. Hoboken, NJ: Wiley (2010).
21. Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., Abrahamsson, P.: Software development in startup companies: A systematic mapping study. *Information and Software Technology*, 56(10), 1200-1218 (2014).
22. Tolvanen, J. P.: *Incremental method engineering with modeling tools: theoretical principles and empirical evidence*. Ph. D. Thesis, University of Jyväskylä (1998).
23. Unterkalmsteiner et al.: Software Startups - A Research Agenda. *E-Informatica Software Engineering Journal*, 1, 89-124 (2016).



### III

## **ECCOLA: A METHOD FOR IMPLEMENTING ETHICALLY ALIGNED AI SYSTEMS**

by

Ville Vakkuri, Kai-Kristian Kemell & Pekka Abrahamsson, 2020

*46th Euromicro Conference on Software Engineering and Advanced Applications  
(SEAA) (pp. 195-204). IEEE*

DOI 10.1109/SEAA51224.2020.00043

Reproduced with kind permission by IEEE.

## ECCOLA - a Method for Implementing Ethically Aligned AI Systems

Ville Vakkuri [0000-0002-1550-1110]  
Faculty of Information Technology  
University of Jyväskylä  
Jyväskylä, Finland  
ville.vakkuri@jyu.fi

Kai-Kristian Kemell [0000-0002-0225-4560]  
Faculty of Information Technology  
University of Jyväskylä  
Jyväskylä, Finland  
kai-kristian.o.kemell@jyu.fi

Pekka Abrahamsson [0000-0002-4360-2226]  
Faculty of Information Technology  
University of Jyväskylä  
Jyväskylä, Finland  
pekka.abrahamsson@jyu.fi

**Abstract**— Various recent Artificial Intelligence (AI) system failures, some of which have made the global headlines, have highlighted issues in these systems. These failures have resulted in calls for more ethical AI systems that better take into account their effects on various stakeholders. However, implementing AI ethics into practice is still an on-going challenge. High-level guidelines for doing so exist, devised by governments and private organizations alike, but lack practicality for developers. To address this issue, in this paper, we present a method for implementing AI ethics. The method, ECCOLA, has been iteratively developed using a cyclical action design research approach. The method aims at making the high-level AI ethics principles more practical, making it possible for developers to more easily implement them in practice.

**Keywords**—Artificial Intelligence, AI ethics, Ethics, implementing, method

### 1. INTRODUCTION

As we make increasing progress on Artificial Intelligence (AI), the systems become increasingly widespread and exert a growing impact on society. This has also resulted in us witnessing various AI system failures, which have served to highlight various ethical issues associated with these systems. Many of these failures have made the global headlines and resulted in public backlash. Especially privacy issues related to facial recognition technology have become a prominent topic among the general public, as well as for policymakers.

The systems we develop, despite us having had some collective learning experiences from past system failures, are still far from being problem-free. Ethical issues persist, and more arise as the technologies become more sophisticated. Aside from the obvious physical damage potential of systems such as autonomous vehicles, data handling alone is ripe with ethical issues without universal answers.

The discussion on the field of AI ethics has soared in activity in the past decade following this technological progress, resulting in the birth of some key principles that are now widely acknowledged as central issues in AI ethics. One such issue is the demand for AI systems that are explainable [1]. The problem thus far has been transferring this discussion into practice. I.e., how to actually influence the development of these systems?

For the time being, this has mostly been carried out either via guidelines or laws and regulations. Guidelines have been devised by companies [2], governments [3] and standardization organizations [4]. Yet, these guidelines have been lacking in actionability. Developers struggle to implement abstract ethical guidelines into the development process [5,6].

Methods and practices in the area remain highly technical, focusing on specific issues in e.g. machine learning [7]. While certainly useful in their specific contexts, these types of tools do not help companies in the design and development process as a whole. Thus, development methods are still required to bridge this gap between research and practice in the area.

In this paper, we present our work on an AI ethics method: ECCOLA. ECCOLA has been developed iteratively over the past two years through empirical use and data resulting from it, with each iteration improving the method. ECCOLA is intended to help organizations implement AI ethics in practice, in an actionable manner.

The rest of this paper is structured as follows. The second section discusses the theoretical background of the paper: AI ethics, methods in AI ethics, as well as the Essence Theory of Software Engineering used in devising the method in question. The third section presents the method, ECCOLA. In the fourth section we discuss how ECCOLA was iteratively developed and what kind of data were used in doing so. In the fifth and final section we discuss the method in relation to extant literature and conclude the paper.

### 2. THEORETICAL BACKGROUND

This section is split into three subchapters. In the first one, we provide an overview of the current state of AI ethics in research. In the second one, we focus on the state of the practical implementation of AI ethics, discussing the methods and other tools that currently exist to help practitioners implement it. In the third and final one, we discuss the Essence Theory of Software Engineering, and specifically the idea of essentializing software engineering practices, as this an approach we have utilized in devising ECCOLA.

#### A. AI Ethics

AI ethics is a long-standing area of research. In the past, much of the debate focused on hypothetical future scenarios that would result from technological progress.

However, as these hypothetical future scenarios start to become reality following said progress, which to many has been faster than anticipated, the field has become increasingly active.

Much of the research in the area has focused on theory, and specifically to define AI ethics by highlighting key ethical issues in AI systems. This discussion has focused on principles. Many have been proposed and discussed, and, by now, some have become largely agreed-upon [8]. Based on an analysis of the numerous AI ethics guidelines that now exist, Jobin et al. [8] listed the key principles that could be considered central based on how often they appear in these guidelines: "transparency, justice and fairness, non-maleficence, responsibility, privacy, beneficence, freedom and autonomy, trust, dignity, sustainability, and solidarity."

To provide an example of the type of research that has been conducted on these principles, we can look at transparency. Transparency [9] is widely considered one of the central AI ethical principles. Transparency is about understanding AI systems, how they work, and how they were developed [9,10]. It has been argued to be the very foundation of AI ethics: if we cannot understand how the systems work, we cannot make them ethical either [11]. The discussion on transparency has, aside from defining what it is, focused on how to achieve it. For example, Ananny & Crawford [10] discussed the limitations of the idea of transparency in relation to the complexity brought on by machine learning. Is being able to see inside the system really enough or even helpful? Transparency is featured as a key principle in the high-profile guidelines of EU [3] and IEEE [4], for example.

Though principles are one way of categorizing the discussion in the area, it is ultimately about bringing attention to potential ethical issues in AI, with or without pinning them under a specific principle. Privacy issues, for example, have been one prominent topic of discussion both in academia and the media following various practical examples of (ethical) AI system failures. Privacy issues have been discussed in relation to data handling, technology such as facial recognition, as well as racial bias, which falls under the principle of fairness.

Indeed, guidelines have, thus far, been the main way of bridging the gap between research and practice in the area. The purpose of these guidelines has been to distill the discussion in the area into a tool. However, past research has shown that guidelines are rarely effective in software engineering. McNamara et al. [6] studied the impact the ACM Code of Ethics<sup>1</sup> had had on practice in the area, finding little to none. This seems to also be the case in AI ethics: in a recent paper [5], we studied the current state of practice in AI ethics and found that the principles present in literature are not actively tackled out on the field.

---

<sup>1</sup> <https://www.acm.org/code-of-ethics>

This state of affairs underlines a need for more actionable tools for implementing AI ethics in practice. In the context of software engineering, we thus turn to methods, i.e. ways of working that direct how work is carried out [12]. As software engineering in any organization is carried out using typically some form of an agile method [13], hybrid or in-house ones, incorporating AI ethics into these methods would be a goal to strive for.

## B. Methods in AI Ethics

There are already various methods and tools for implementing AI ethics, as highlighted by Morley et al. [7] in their systematic review. These are largely tools for the technical side of AI system development, such as tools for machine learning. On the other hand, we are not currently aware of any method focusing on the higher-level design and development decisions surrounding AI systems. Guidelines have been devised for this but seem to remain impractical given their seeming lack of adoption out on the field [5].

Aside from AI ethics methods and tools, some ethical tools from other fields do exist that could potentially be used to design ethical AI systems. One example of such a tool is the RESOLVEDD method from the field of business ethics [14]. We have, in a past study [15], studied the suitability of this particular method for the AI ethics context, with our results suggesting that dedicated methods would be more beneficial. Such methods, however, are currently lacking.

Aside from ECCOLA, there is currently some other activity in method development for the area as well, though to the best of our knowledge most of these are still work-in-progress. E.g., while not a software engineering method as such, Leikas et al. [16] recently presented an "ethical framework for designing autonomous intelligent systems". This framework, however, is more focused on higher level design than development and not specifically aimed at developers or product managers.

In devising ECCOLA, our method, we have turned to the Essence Theory of Software Engineering for method engineering. Specifically, we have utilized the theory's philosophy of essentializing software engineering practices in devising a method, as we discuss next.

## C. Essentializing to Create Methods from Practices

The Essence Theory of Software Engineering (Jacobson et al. [12]) is a method engineering tool. It comprises a method core, which the authors refer to as a kernel, as well as a language. The kernel, they argue [12], contains all the core elements present in any software engineering project.

To this end, the kernel contains three types of items: alphas (ie. things to work with), activities (things to do), and competencies (skills required to carry out the tasks). There

are seven alphas, which form the core of the kernel<sup>2</sup>: opportunity, stakeholders, requirements, software system, work, team and way-of-working. The kernel provides a basis for constructing methods using the Essence language to describe them. I.e., the theory consists of basic building blocks which can be utilized by using the language to extend the base to build a method. On its own, the kernel could be used as a generic software engineering method, but the point of Essence is to construct new methods using the language, while utilizing the kernel as an extensible starting point for doing so.

Software engineering methods consist of practices. A practice is a more atomic unit of work, such as pair programming. In creating ECCOLA, we have utilized the idea of essentializing [17] software engineering practices. In short, this refers to describing them using the Essence language. This offers one way of breaking down practices into different elements in order to describe them, making them easier to understand. This also serves to make practices more modular, as describing them in the same notational language makes it easier to combine them into methods.

Essentializing practices is described as a process by Jacobson [17] as follows:

*"- Identifying the elements – this is primarily identifying a list of elements that make up a practice. The output is essentially a diagram [...]*

*- Drafting the relationships between the elements and the outline of each element – At this point, the cards are created.*

*- Providing further details – Usually, the cards will be supplemented with additional guidelines, hints and tips, examples, and references to other resources, such as articles and books"*

As can be observed in the above quote, Essence utilizes cards to describe methods. This is also an approach we have utilized in ECCOLA: ECCOLA is a card deck.

Essence was also chosen due to its method-agnostic approach and modular philosophy on methods. From the get-go, ECCOLA was never intended to be a stand-alone method, but rather, a modular extension to existing software development methods that would bring in AI ethics.

Originally, we planned on using the Essence language to describe ECCOLA. For example, principles such as transparency could have been alphas (i.e. things to work with) in the method. However, as the development of the method progressed and we began to test its early versions in practice, Essence turned out to make the method confusing to its users. As a result, the role of Essence in ECCOLA grew smaller, as we discuss in the fourth section.

### 3. ECCOLA - A METHOD FOR DESIGNING ETHICALLY ALIGNED AI SYSTEMS

As we have discussed in section II, AI ethics is currently an area with a prominent gap between research and

practice. Much of the research has been theoretical and conceptual, focusing on defining key principles for AI ethics and how to tackle them. The numerous guidelines for AI ethics that currently exist [8] have tried to bridge this gap to bring these principles to the developers, but seem to not have had much success. Indeed, ethical guidelines tend to not have much impact in the context of SE [6]. To bridge this gap we propose a method for implementing AI ethics: ECCOLA.

ECCOLA<sup>3</sup> (figure 1) is intended to provide developers an actionable tool for implementing AI ethics. To utilize the various AI ethics guidelines in practice, the organization seeking to do so has to somehow make them practical first. ECCOLA, on the other hand, is intended to be practical as is, and ready to be incorporated into any existing method. ECCOLA does not provide any direct answers to ethical problems, as arguably correct answers are a rare breed in ethics in general, but rather asks questions in order to make the organization consider the various ethical issues present in AI systems. Though ultimately how these questions are then tackled is up to the organization in question, ECCOLA does encourage taking into account any ethical issues it highlights.

ECCOLA is built on AI ethics research. It utilizes both existing theoretical and conceptual research, as well as AI ethics guidelines that have been devised based on existing research as well. In terms of guidelines, the cards are based primarily on the IEEE Ethically Aligned Design guidelines [4] and the EU Trustworthy AI guidelines [4]. As these guidelines have already distilled much of the existing research on the topic under various principles, these principles have been utilized in ECCOLA as well. AI ethics research has been used to further expand on these principles in ECCOLA.

In practice, ECCOLA takes on a form of a deck of cards. This approach was based on the Essence Theory of Software Engineering [12], which was used to describe the first versions of the method. Methods described using the Essence language are utilized through cards. However, using cards in the context of software engineering methods is not a novel idea, nor one proposed by Essence. E.g., Planning Poker in Agile uses cards and the idea of Kanban is founded around using cards in the form of sticky notes.

There are 21 cards in total in ECCOLA. These cards are split into 8 themes, with each theme consisting of 1 to 6 cards. These themes are AI ethics themes found in various ethical guidelines [8], such as transparency or data. Each individual card, then, deals with a more atomic aspect of that theme, such as, in the case of data, data privacy and data quality. Aside from the main set of cards, ECCOLA also features an A5-sized game sheet that describes how the method is used.

<sup>2</sup> <http://semat.org/alpha-definitions-overview/competency-cards>

<sup>3</sup> [https://figshare.com/articles/Internet\\_resource\\_for\\_ECCOLA\\_-\\_a\\_Method\\_for\\_Implementing\\_Ethically\\_Aligned\\_AI\\_Systems/12136308](https://figshare.com/articles/Internet_resource_for_ECCOLA_-_a_Method_for_Implementing_Ethically_Aligned_AI_Systems/12136308)



This is the author's version of the work. The definite version was published in: V. Vakkuri, K. -K. Kemell and P. Abrahamsson, "ECCOLA - a Method for Implementing Ethically Aligned AI Systems," 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 2020, pp. 195-204, <https://doi.org/10.1109/SEAA51224.2020.00043>

# ECCOLA

### Game Sheet – How to Play the Cards

**Info:** ECCOLA is easy to apply in practice. It is a sprint-by-sprint evolving process that empowers ethical thinking in the product development process. As a result, development is enhanced and Work Product Sheets (WPS) are created. The WPSs help you measure the Trustworthiness of the product. ECCOLA is an evolving set of cards and you choose the cards that are relevant to your work.

**How to:** ECCOLA is intended to be used during the entire design and development process in three steps:

1. Prepare - Choose the relevant cards for the current sprint. Document selected cards and justification on WPS.
2. Review - Keep the selected cards on hand during single tasks. Write down if any actions are taken based on the cards.
3. Evaluate - Review to ensure that all planned actions are taken. Revise the card card and if necessary, review tasks again.

**Practical Tip:** Repeat the process in every iteration. Remember to do a retrospective afterwards. Think about what worked & what did not. Choose the parts that are the most relevant for your work in the next round.

### #0 Stakeholder Analysis

**Motivation:** In order to understand the big picture, it is important to first understand who the system can affect and how. Try to also think past the obvious, direct stakeholders such as your end-users.

**What to Do:** Identify stakeholders.

- Who does the system affect, and how? Stakeholders are not simply users, developers and customers.
- Can these different stakeholders influence the development of the system? How?
- Remember that a user is often an organization and the end-user is an individual. Similarly, AI systems can treat people as objects for data collection.

**Practical Example:** Autonomous cars don't just affect their passengers. Anyone nearby is affected, some even change the way they drive. If at one point half of the traffic consists of self-driving cars, what are the societal impacts of such systems? E.g., how are the people who can't afford one affected? Regulations arising from such systems also affect everyone.

### #1 Types of Transparency

**Motivation:** When considering transparency, it is important to understand who you are being transparent towards, and what you are being transparent about.

**What to Do:** Consider the following...

- Are you trying to understand something? (Internal transparency)
- Are you trying to explain something? (External transparency)
- Are you trying to understand or explain how the system works? (Transparency of algorithms and data)
- Are you trying to understand or explain why the system was made to be the way it is? (Transparency of system development)
- External stakeholders to consider, among others: (end-users, safety certification agencies, accident investigators, lawyers or expert witnesses, and society at large for disruptive technologies)

### #2 Explainability

**Motivation:** If we cannot understand the reasons behind the actions of the AI, it is difficult to trust it.

**What to Do:** Ask yourself:

- Is explainability a goal for your system? How do you plan to ensure it?
- How well can each decision of the system be understood? (by both developers and end-users)
- Did you try to use the simplest and most interpretable model possible for the context?
- Did you make trade-offs between explainability and accuracy? What kind of? Why?
- How familiar are you with your training or testing data? Can you change it when needed?
- If you utilize third party components in the system, how well do you understand them?

**Practical Example:** When interacting with a robot, users could clearly ask the robot "why did you do that?" and receive an understandable response. This would make it much easier for them to trust a system.

### #6 System Reliability

**Motivation:** Transparency makes ethical development possible in the first place. To make it reliable, we must understand how the system works and why it makes certain decisions.

**What to Do:** Ask yourself:

- How do you test if the system fulfills its goals?
- How do you test the system comprehensively, including unlikely scenarios? Have the tests been documented?
- When the system fails in a certain scenario, will you be able to tell why? Can you replicate the failure?
- How do you assure the (end-)user of the system's reliability?

**Practical Example:** An autonomous coffee machine successfully brews coffee 8 times out of 10. While this is a decent success rate, we are left wondering what happened the 2 times it failed to do so, and why. Errors are inevitable, but we must understand the causes behind them and be able to replicate them to fix them.

### #5 Traceability

**Motivation:** Traceability supports explainability. It helps us understand why the AI acts the way it does.

**What to Do:** Document. Different types of documentation (code, project etc.) are typically key in producing transparency.

- How have you documented the development of the system, both in terms of code and decision-making? How was the model built or the AI trained?
- How have you documented the testing and validation process? In terms of data and scenarios used etc.
- How do you document the actions of the system? What about alternate actions (e.g. if the user was different but the situation otherwise the same)?

**Practical Example:** When the system starts making mistakes, by means of traceability, it will be easier to find out the cause. Consequently, it will also be faster and possibly easier to start fixing the underlying issue.

### #4 Documenting Trade-offs

**Motivation:** One important part of choosing a transparent system development is the documentation of trade-offs. Whenever you make a decision, you choose one option over other alternatives. However, documenting why and what the alternatives were is important.

**What to Do:** Ask yourself:

- Are relevant interests and values impacted by the system and potential trade-offs between them identified and documented?
- Who decides on such trade-offs (e.g. between two competing solutions) and how? Did you ensure that the trade-off options and the reasons behind them were documented?

**Practical Example:** Documenting trade-offs can improve your customer relationship, allowing you to better explain why certain decisions were made over others. Moreover, it can reduce the responsibility placed on the individual developer(s) from an ethical point of view.

### #3 Communication

**Motivation:** In practice, communication is a big part of being transparent with your stakeholders. Being transparent in communication can generate trust.

**What to Do:** Ask yourself:

- What is the goal of the system? Why is this particular system deployed in this specific area?
- What do you communicate about the system to its users and end-users? Is it enough for them to understand how the system works?
- If relevant to your system, do you somehow tell your (end-)users that they are interacting with an AI system and not with another human being?
- Do you collect user feedback? How is it used to change/improve the system?

**Practical Example:** Clear communication and transparency towards other audiences, such as the general public, increases trust.

### #7 Privacy and Data

**Motivation:** Privacy is a rising trend in the wake of various recent data misuse revelations. People are now increasingly conscious about handling their personal data. Similarly, regulations such as the GDPR now affect data collection.

**What to Do:** Ask yourself:

- What data are used by the system?
- Does the system use or collect personal data? Why? How is the personal data used?
- Do you clearly inform your (end-)users about any personal data collection? (E.g., ask for consent, provide an opportunity to revoke it, etc.)
- How have you taken measures to enhance (end-user) privacy, such as encryption or anonymization?
- Who makes the decisions regarding data use and collection? Do you have organizational policies for it?

**Practical Example:** Rather than collecting and selling data, appearing to privacy can also be profitable. Regulations are making it increasingly difficult to collect lots of personal data for profit. Privacy can be an alternate selling point in today's climate.

### #8 Data Quality

**Motivation:** As AI are trained using data, the data used directly affects how the system operates. Both the nature and the quality and integrity of the data used will align with the goals of the system.

**What to Do:** Ask yourself:

- What are good or poor quality data in the context of your system?
- How do you evaluate the quality and integrity of your own data? Are there alternative ways?
- If you utilize data from external sources, how do you control their quality?
- Did you align your system with relevant standards (for example ISO 9001 or similar) or develop protocols for daily data management and governance?
- How can you tell if your data sets have been hacked or otherwise compromised?

**Practical Example:** In 2017, Amazon scrapped its recruitment AI because of bias data. They used past recruitment data to teach the AI. As they had mostly hired men, the AI began to consider women undesirable based on the data.

### #9 Access to Data

**Motivation:** Aside from carefully planning what data you collect and how, it is also important to plan how it can or will be used and by whom.

**What to Do:** Ask yourself:

- Who can access the users' data, and under what circumstances?
- How do you ensure that the people who access the data: 1) have a valid reason to do so, and 2) adhere to the regulations and policies related to the data?
- Do you keep logs of who accesses the data and when? Do the logs also tell why?
- Do you use existing data governance frameworks or protocols? Does your organization have its own?
- Who handles the data collection, storage, and use?

**Practical Example:** Third parties you give access to the data can misuse it. A prominent example of this is the data from Facebook used to train a recommendation AI. How should data from Facebook be used questionably. However, such incidents can also point your organization in a bad direction.

### #10 Human Agency

**Motivation:** People interacting with the system or using it should be able to understand it sufficiently. Users should be able to make informed decisions based on its suggestions, or to challenge its suggestions. AI systems should let humans make independent choices.

**What to Do:** Ask yourself:

- Does the system interact with decisions by human action, i.e. end users (e.g. recommending users action or decisions, or presenting options)?
- Does the system communicate to its (end) users that a decision, content or outcome is the result of an algorithmic decision? Into how much detail does it go?
- In the system's use context, what tasks are done by the system and what tasks are done by humans?
- Have you taken measures to prevent overconfidence or overreliance on the system?

**Practical Example:** A medical system recommends diagnoses. How does the system communicate to doctors why it made a recommendation? How should the doctors know when to challenge the system? Does the system somehow change how patients and doctors interact?

### #11 Human Oversight

**Motivation:** AI systems should support human decision-making. They should not undermine human autonomy by making decisions for us, meaning they should be subject to human oversight.

**What to Do:** Ask yourself:

- Who can control the system and how? In what situations?
- What would be the appropriate level of human control for this particular system and its use cases?
- Related to the safety and security cards: how do you detect and respond if something goes wrong? Does the system then stop entirely, partially, or would control be delegated to a human? Why?

**Practical Example:** Assuming control to especially related to cyber-physical systems such as drones or other vehicles. For purely digital systems, the focus should be on supporting human decision-making instead of directing.

### #12 System Security

**Motivation:** While cybersecurity is important in any system, AI systems present new challenges. Cyber-physical systems can even cause fatalities in the hands of malicious actors.

**What to Do:** Ask yourself:

- What are potential forms of attacks to which the system is vulnerable? Did you consider ones that are unique or more relevant to AI systems?
- Did you consider different types of vulnerabilities, such as data pollution or model poisoning, or how do you detect and respond if something goes wrong? Does the system then stop entirely, partially, or would control be delegated to a human? Why?
- Does your organization have cybersecurity personnel? Are they involved in this system?

**Practical Example:** The autonomous nature of AI systems makes new vectors of attack possible. A white line drawn across a road can cause a self-driving vehicle. What happened to Microsoft's Tay Twitter bot is another example of a new type of attack.

### #13 System Safety

**Motivation:** AI systems exert notable influence on the physical world whether they are cyber-physical or not. Various risks and their consequences should be considered, thinking ahead to the operational life of the system.

**What to Do:** Ask yourself:

- What kind of risks does the system involve? What kind of damage could it cause?
- How do you measure and assess risks and safety?
- What without plans does your system have? How have they been tested?
- What conditions do the failure plans trigger? How are scenarios or do they require human input?
- Is there a plan to mitigate or manage technological errors, accidents, or malicious misuse? What if the system provides wrong results, becomes unavailable, or provides constantly unacceptable results?
- What liability and consumer protection laws apply to your system? How are claims from users assessed?

**Practical Example:** AI systems can aid automating various organizational tasks, making it possible to reduce personnel. However, if a customer organization becomes reliant on your AI system to handle a portion of its operations, what happens if that AI stops functioning for even a few days? What could you do to alleviate the impact?

### #14 Accessibility

**Motivation:** Technology can be discriminating in various ways. Given the enormous impact AI systems can have, ensuring equal access to their positive impacts is ethically important.

**What to Do:** Ask yourself:

- Does the system consider a wide range of individual preferences and abilities? If not, why?
- Is the system usable by those with specific needs or disabilities? In the event of exclusion, or those using assistive technologies?
- Were people representing various groups somehow involved in the development of the system?
- How is the potential user audience taken into account? Is the user involved in building the system representative of your target user audience? Is it representative of the general population?
- Did you assess whether there could be (groups of) people that are already technologically capable, resulting in increased inequality? E.g. most of the images used in machine learning have been labeled by young white men.

### #15 Stakeholder Participation

**Motivation:** As AI systems have notable impacts, their stakeholders are also numerous. Though the system affects these various holders in various ways, they are often not involved in the development. Yet, e.g. when using a decision-making system, its users have to trust the system while also being critical of it.

**What to Do:** Turn to your stakeholder analysis (Card #0):

- Which stakeholders are stakeholders in system development?
- How are the different stakeholders of the system involved in the development of the system? (If they aren't, why?)
- How do you inform your external and internal stakeholders of the system's development?

**Practical Example:** Often the people an AI system is used on are individuals who are only objects for the system. For example, a medical system is developed for hospitals, used by doctors, but ultimately used on patients. Why not talk to the patients too?

### #16 Environmental Impact

**Motivation:** Past the general wellbeing implications, ecological consciousness is a current trend. Being ecological can be a selling point for your organization.

**What to Do:** Ask yourself:

- Do you assess the environmental impact of the system's development, deployment, and use? (e.g. the energy used by the data centers)
- Do you consider the environmental impact when selecting specific technical solutions?
- Did you ensure measures to reduce the environmental impact of your system's life cycle?

**Practical Example:** If you are hosting on a third party cloud, try to assess the sustainability of the service provider's services. If you are using hardware, are you processing the data in each physical device of your own or are you processing it in the cloud?

### #17 Societal Effects

**Motivation:** The impacts a system has go beyond its users. A system may well affect negatively even those who do not use it (or wish to use it).

**What to Do:** Ask yourself:

- Did you assess the broader societal impact of the AI system beyond the individual (end-)user? Consider stakeholders who might be indirectly affected by the system.
- How will the system affect society when in use?
- What kind of systemic effects could the system have?

**Practical Example:** Surveillance technology utilizing facial recognition AI has long-reaching impacts. People may wish to avoid areas that utilize such surveillance, negatively affecting businesses in said areas. People may become stressed at the mere thought of such surveillance. Some may even emigrate as a result.

### #18 Auditability

**Motivation:** Regulations affecting AI and data may necessitate audits of systems in the future. Similarly, if the system causes damage, an audit might be requested. It is good to have mechanisms in place beforehand.

**What to Do:** Ask yourself:

- Is the system auditable?
- Can an audit be conducted independently?
- Is the system available for inspection?
- What mechanisms facilitate the system's auditability? How is (in)visibility and logging of the system's processes and outcomes ensured?

**Practical Example:** In heavily regulated fields such as medicine, audits are typically required before a system can be utilized in the first place.

### #19 Ability to Redress

**Motivation:** Making sure people know they can be compensated in some way in the event something goes wrong with the system is important in generating trust. Such scenarios should be planned in advance to what extent possible.

**What to Do:** Ask yourself:

- What is your (developer/organization) responsibility if the system causes damage or otherwise has a negative impact?
- In the event of negative impact, can the one affected seek redress?
- How do you inform users and other third parties about opportunities for redress?

**Practical Example:** AI systems can inconvenience users in unforeseen, unpredictable ways. Depending on the situation, the company may or may not be legally responsible for the inconvenience. Nonetheless, by offering a digital platform for seeking redress, your company can seem more trustworthy while also offering additional value to your users.

### #20 Minimizing Negative Impacts

**Motivation:** Minimizing negative impacts of the system is financially important for any developer organization. Incidents are often costly.

**What to Do:**

- First, consider...
  - Is your stakeholder analysis up-to-date (Card #0)
  - Have you discussed IR&D (Card #1)
  - Have you discussed auditability? (Card #18)
  - Have you discussed redress issues? (Card #19)
- Are the people involved with the development of the system also involved with it during its operational life? If not, why not? Can you involve them?
- Are you aware of laws related to the system?
  - Can users of the system somehow report vulnerabilities, bugs, and other issues (to a system)?
  - With whom have you discussed accountability and other ethical

## Card Themes

Analyze  
Transparency  
Safety & Security  
Fairness

Data  
Agency & Oversight  
Wellbeing  
Accountability

Ville Vakkuri [villvakkuri@jyu.fi](mailto:villvakkuri@jyu.fi) Kai-Kristian Kemell [kai-kristian.o.kemell@jyu.fi](mailto:kai-kristian.o.kemell@jyu.fi)  
 Pekka Abrahamsson [pekka.abrahamsson@jyu.fi](mailto:pekka.abrahamsson@jyu.fi)

Figure 1. ECCOLA - a Method for Implementing Ethically Aligned AI Systems

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Each card in ECCOLA is split into three parts (figure 2): (1) motivation (i.e. why this is important), (2) what to do (to tackle this issue), and (3) a practical example of the topic (to make the issues more tangible). Each card also comes with a note-making space. As the cards are generally utilized as physical cards, the card is split into two with the left half of each card containing the textual contents and the right half containing white space for notes. This note-making space has been included to make using the cards more convenient in practice.

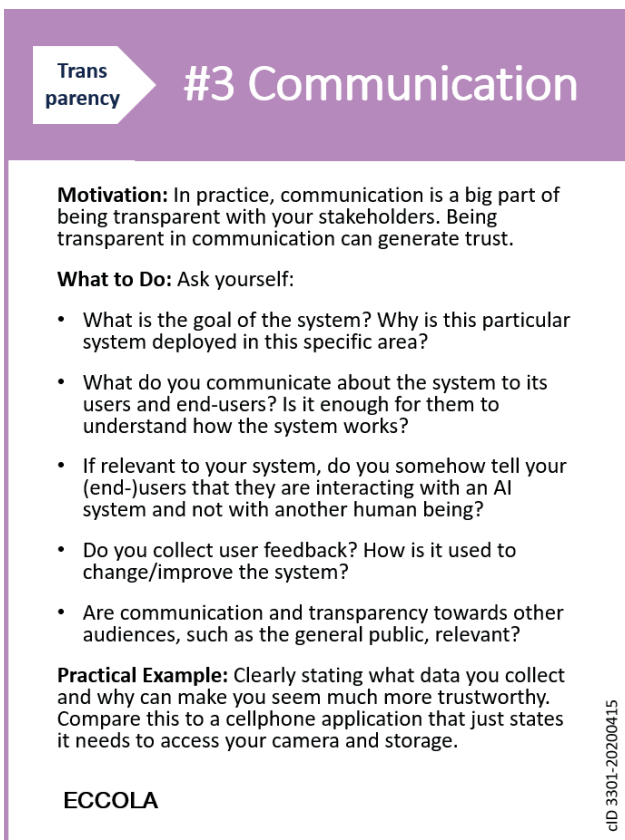


Figure 2. Card example from ECCOLA, Card #3 Communication

ECCOLA supports iterative development. During each iteration, the team is to choose which cards, or themes, are relevant for that particular iteration. ECCOLA is also method-agnostic, making it possible to utilize it with any existing or in-house SE method.

Depending on by whom ECCOLA is utilized, the tool has different goals. First, for product owners, the tool is intended to result in non-functional user stories involving ethics. Secondly, for a team of developers, the goal of ECCOLA is facilitating communication. By using the cards, the team will end up discussing ethical issues and making decisions based on the discussions. Finally, if utilized by a single developer, the goal of the method is raising awareness of ethical issues in AI. A single developer would instead dwell on these potential issues on their own while

possibly looking further into the issues online for other points of view.

In developing ECCOLA, we have had three main goals for the method:

- To help create awareness of AI ethics and its importance
- To make an adaptable, modular method suitable for a wide variety of SE contexts, and
- To make ECCOLA suitable for agile development, and to also make ethics a part of agile development in general.

Next, we discuss how ECCOLA has been developed. It has been developed iteratively with multiple sets of data.

#### 4. ECCOLA DEVELOPMENT PHASES AND DATA

ECCOLA has been developed iteratively through multiple phases (five, thus far). For this purpose, we have utilized the Cyclical Action Research method described by Susman and Evered [18] in developing it. In each phase, we have collected empirical data, based on which ECCOLA has been improved (figure 3).

The subsections of this section each cover one phase. In each subsection, we discuss what ECCOLA looked like at the time, how it was tested, and how it was changed based on the data. This process is also summarized in Table 1.

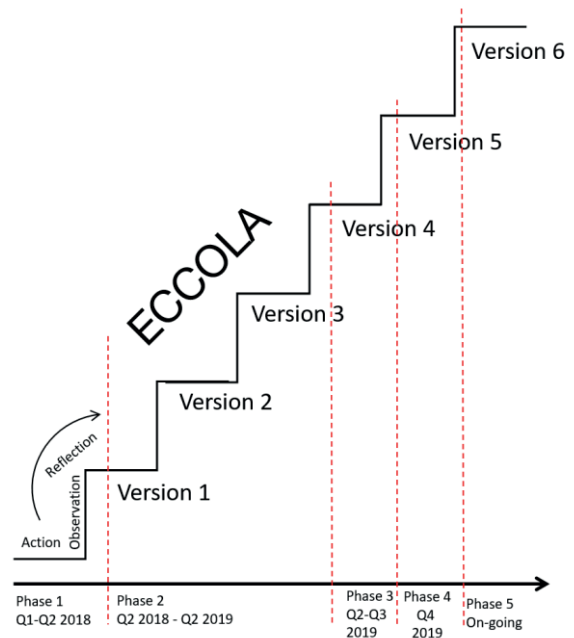


Figure 3. Cyclical Action Research process on ECCOLA. Including Cycle of Action, Observation, Reflection on each iteration

##### A. Phase 1 (Q1-Q2 2018)

In early 2018, prior to starting our work on ECCOLA, we searched for existing methods for AI ethics, ultimately

finding none. Thus, we expanded our horizons and looked at ethical tools from other fields instead, to see if anything would seem applicable in the context of AI ethics as well. This led us to eventually test an existing ethical tool from the field of business ethics, the RESOLVEDD strategy [14], in the context of AI ethics. Our aim was to see if existing ethical tools, even if they were not specifically created for AI ethics, could be suitable for that context.

We conducted a scientific study on RESOLVEDD in the context of AI ethics. These findings have been published in-depth elsewhere (see Vakkuri & Kemell [15]). In short, we discovered that forcing developers to utilize RESOLVEDD did have some positive effects. Namely, it produced transparency in the development process, and the presence of an ethical tool made the developers aware of the potential importance of ethics, resulting in ethics-related discussions within the teams. However, the tool itself was not considered well-suited for the context by the respondents. Moreover, when forcing developers to utilize such a tool, the commitment towards it quickly vanished when the tool was no longer compulsive.

half of 2018. This initial version of the method was based on three primary theories: (1) RESOLVEDD strategy, (2) The Essence Theory of Software Engineering, and (3) The IEEE Ethically Aligned Design guidelines.

We utilized some of the general ideas of RESOLVEDD, which were deemed useful based on the data we collected. Namely, we took to RESOLVEDD for ideas on how to make the tool support iterative development. Additionally, we included some of the aspects of RESOLVEDD which were shown to support transparency of systems development (e.g. the idea of producing formal text documents while using the method).

We began to describe the method using the Essence language (see section 2.3). Methods described using Essence are visualized through cards, and thus, ECCOLA took on the form of a card deck as well. This also meant that we included the various elements of Essence into the cards. For example, we made some of the key AI ethics principles, namely transparency, accountability, and responsibility, into alphas in the context of Essence (i.e. measurable things to work on). The cards also included various activities that were to be performed in order to progress on these alphas, as well as patterns and other Essence elements.

The AI ethics contents of the method, at this stage, were based primarily on the IEEE Ethically Aligned Design guidelines [4]. We included key principles from the guidelines such as transparency and accountability, which have been prominent topics of discussion in AI ethics. Additionally, we utilized various research articles. For example, to expand on transparency, we utilized the studies of Dignum [9] and Ananny & Crawford [10], among others.

Much like how while using RESOLVEDD one produces text answering some questions posed by the tool, we incorporated the same idea of producing text while using ECCOLA into the initial version of the method. The theoretical background of this early version was based primarily on the IEEE EAD guidelines and the idea of the ART principles of AI Ethics [9].

#### 2) Testing Version 1 (Q1 2019)

This first version of ECCOLA was tested in a large-scale project-based course on systems development at the University of Jyväskylä in the first quarter of 2019. In the course, 27 student teams of 4-5 students worked on a real-world case related to autonomous maritime. Each team was tasked with coming up with an innovation that would help make autonomous maritime possible. The teams were not required to actually develop these innovations into functional products, given the time and capability constraints in a course setting, but rather, to hone the ideas as far as they could in the context of the course. Some teams ultimately did produce technical demos, but this was not required. The results of these projects have been published in an educational book<sup>4</sup>.

TABLE I. CYCLICAL ACTION RESEARCH PHASES

Phase	Version	Primary Background Theories	Study setting	Timing	Study Participants
1	N / A	RESOLVEDD, EAD, Essence	Class	Q1-Q2 2018	5 teams of 4-5 students
2	1	RESOLVEDD, EAD, Essence	Class	Q2 2018 - Q2 2019	27 teams of 3-5 students
	2	RESOLVEDD, EAD, Essence	Class	Q2 2018 - Q2 2019	27 teams of 3-5 students
	3	RESOLVEDD, EAD, Essence	Class	Q2 2018 - Q2 2019	27 teams of 3-5 students
3	4	EU AI HLEG, EAD	Blockchain Project	Q2-Q3 2019	2 sw development team members
4	5	EU AI HLEG, EAD	Conference Workshop	Q4 2019	8 researchers
5	6	EU AI HLEG, EAD	N/A	On-going	N/A

#### B. Phase 2 (Q2 2018 - Q2 2019)

##### 1) Creating Version 1 (Q2 2018 - Q1 2019)

Based on the results of the RESOLVEDD study, we began to develop a method of our own, ECCOLA, during the latter

<sup>4</sup> <https://jyx.jyu.fi/handle/123456789/63051>

As any such innovation would involve AI directly or indirectly, given the autonomous maritime context, we chose to test ECCOLA by having these teams utilize it to reflect on the ethical issues their ideas might pose. The teams were introduced to ECCOLA during a course lecture and were handed a physical card deck. Each team was then told to utilize the card deck in whatever way they saw fit, while writing down notes on the cards as - or if - they used them. Additionally, unstructured interview data was collected from the teams through their weekly meetings with their assigned mentor and this feedback was taken into account in developing the method.

Prior to the course, the students had been tasked with reading a book on Essence, Software Engineering Essentialized [17], which explains the tool. Though the educational goal of this was elsewhere, this also served to make sure the students would not be overtly confused with this version of ECCOLA being described using the Essence language.

After the students had utilized the cards for a week, they were collected and the written notes on them analyzed. Based on this data, and the discussions the teams had had with their mentors in the weekly meetings, ECCOLA was improved as follows. First, alpha states were added to the alphas to make tracking progress on them easier. Secondly, practical examples were added to the cards to make the ethical issues on them more tangible to someone not versed in AI ethics. Thirdly, we improved the language on the cards, reducing academic jargon and focusing on practice. Finally, we removed the academic references that were initially present in each card. These were deemed to provide little value in raising awareness as none of the teams indicated having used them.

### 3) *Testing Version 2, (Q1 2019)*

This iteration took place during the same systems development course described in the preceding subsection. This iteration was carried out in the same manner as the previous one. The same student teams were tasked with utilizing the new version of ECCOLA again while writing down notes on them as they did. Additional data was again collected in the weekly mentor meetings. Overall, this was, in terms of time elapsed, a brief iteration carried out during the course.

After another week, ECCOLA was once more improved based on the data collected. We added a game sheet describing how the cards and the method should be used. This was done because it became clear that we had to teach the users of the method to use it as it lacked clear instructions. The cards were also numbered to make the method easier to grasp and to make it easier for the cards to refer to each other. To this end, we also improved the language on the cards, aiming to reduce academic jargon.

### 4) *Testing Version 3 (Q1 2019)*

As was the case with the previous two iterations in this phase, the third version of ECCOLA was tested in the

systems development course in a similar manner. However, as this was towards the end of the course, there were no further iterations to be tested in the same setting. Thus, we took our time to analyze the feedback from all three versions, reflect on it, and study new publications in the area to improve the method.

This resulted in a lengthier creation process for the subsequent version. Based on the data and our reflection we made larger changes to the method. We discuss these in the following subsection.

### 5) *Creating Version 4 (Q2 2019)*

Data from phase 2 indicated that the method, though cumbersome to use, did help the teams implement AI ethics. The notes they had made on the cards showed that they had conducted ethical analyses successfully and changed their ideas based on their analyses. The AI ethics portion of the method thus worked. However, the method was not easy to use.

After the course had concluded, we had time to make larger improvements to the method based on the data. We opted to lessen the role of Essence in the method, forgoing the idea of using the Essence language to describe it. It seemed that Essence had made ECCOLA more confusing than it otherwise would have been, as in addition to learning the method, its users would have to learn the Essence notation and Essence in general. We stopped using the Essence elements in the cards and instead split the cards into different AI ethics themes. However, the general approach of using cards for the method seemed to work and thus this approach was kept.

The role of Essence in ECCOLA remains largely in relation to the idea of essentializing practices. This is described in the quote in section II C. ECCOLA aims to distill the essential parts of the AI ethics principles in the guidelines while making them more actionable through the card format.

Additionally, based on the data, the method seemed to be too heavy to use. ECCOLA was initially designed to be a linear process that was iteratively repeated. Its users, however, would be free to modify the process based on their development context and based on their use experience. Nonetheless, this approach was considered too rigid, and the respondents felt it was just another process tacked onto their other work processes. Moreover, the teams were using the method in a modular fashion, using individual cards as they deemed suitable, despite the instructions telling them to use it as a process.

We thus changed the approach, making the cards more stand-alone. In doing so, we wanted to make ECCOLA more modular by design, so that the users of the method could indeed choose which cards to utilize based on which ones they felt were relevant for their current situation. We felt that this would also make ECCOLA easier to use in conjunction with other methods.

During this time period, before the next empirical test, we also expanded the theoretical basis of the method. The

initial version of the EU Guidelines for Trustworthy AI were published in early 2019, some aspects of which we chose to incorporate into ECCOLA. Other novel literature was also included to expand on theoretical basis of the method.

### C. Phase 3 (Q2-Q3 2019)

As the primary concern with the versions 1-3 had been the way ECCOLA was used as a method in practice rather than its AI ethical contents, we chose to focus on making a method that is easier and more practical to use. For this purpose, we made a spin-off of ECCOLA for the context of blockchain ethics. Many of the AI ethical themes such as transparency and data issues could be translated into this context, even if the contents of the cards had to be modified to be better suited for it. Additional blockchain specific issues were also added into these cards.

In this phase, ECCOLA was utilized in a real-world blockchain project by two of the project team members. Data was collected through observation and various unstructured interviews. The team was free to utilize the cards as they wished, and was encouraged to reflect on how the method would best suit their SE development method of choice. However, the team could also receive consultation from one of the researchers where needed on how to use the cards, as well for clarification on their contents, if needed. As a result, we gained a better understanding of how the method was utilized in practice (e.g., how many cards were used per iteration on average, which was 6) in a real-world SE context.

Notably, in this phase, ECCOLA was utilized in conjunction with existing SE methods, namely SCRUM. The feedback regarding the use of ECCOLA with another method was positive, lending support to the idea that ECCOLA does work as a modular method, especially with Agile methods. However, more testing is still needed in this regard in the future.

Based on the data gathered from the blockchain project, the main ECCOLA card deck was iteratively improved. The lessons learned from studying the use of the blockchain ethics version of ECCOLA were incorporated into ECCOLA. The data from this phase was primarily used to improve the contents of the cards by adding more contextual content (i.e. why these things are important) into each card. In this phase, the cards were also split into themes for clarity of presentation. Finally, stakeholder analysis was deemed to require more focus based on the data, and thus cards to support it were added.

### D. Phase 4 (Q4 2019)

After improving ECCOLA based on the lessons learned from the blockchain project, we presented ECCOLA at the 10th International Conference on Software Business, ICSOB2019<sup>5</sup>, in a workshop. In the workshop the

participants utilized ECCOLA to discover potential ethical issues in a given, hypothetical AI development scenario. The participants of the workshop were split into two groups for the task.

The first group was tasked with developing an idea for an AI-based drone that would help farmers improve their harvests. The second group was tasked with developing an AI-based system that would filter and evaluate immigration applications. During the workshop, the groups worked on the ideas iteratively in timed sessions. Each group had a customer stakeholder that progressively presented them with more requirements at the end of each iteration. For every iteration, the groups were to select the cards they felt would be most relevant for the requirements of that iteration.

At the end of the workshop, verbal feedback from the participants was collected. This was done in the form of a discussion where the participants talked about their experiences with each other and between the two groups. These group interviews were recorded and later transcribed for analysis.

The feedback was then utilized to develop the current version of ECCOLA. The themes were color coded for further clarity of presentation. Additionally, we expanded the motivation and practical example portions of some of the cards to make them more stand-alone. E.g., in some cases, a user might have had to search online for more information on some past incident that was only mentioned by name.

### E. Phase 5 (On-going)

The development of ECCOLA continues. We argue that we have now reached a stage of maturity where ECCOLA can be brought forward to the scientific community. However, the method is not finalized and its development and testing continues in this iterative manner. The current version of ECCOLA, discussed in this paper, will again be tested and iteratively improved in the future (The most recent version is available at [bit.ly/eccola-for-ai-ethics](http://bit.ly/eccola-for-ai-ethics)).

However, we feel that we have now reached a point of maturity where we wish to share the method with the scientific community. We discuss our reflections on the current state of ECCOLA in the next and final section of the paper in detail.

## 5. DISCUSSION AND CONCLUSIONS

In this paper, we have presented a method for implementing AI ethics: ECCOLA. ECCOLA is intended to help organizations develop more ethical AI systems by providing them with means of implementing AI ethics in a practical manner. ECCOLA has been developed iteratively using the Cyclical Action Research approach [18]. Though development on the method continues, we have reached a state of maturity where we want to share the method with the scientific community.

<sup>5</sup> <https://icsob2019.wordpress.com/workshops/>

The purpose of ECCOLA is to help us bridge the gap between research and practice in the area of AI ethics. Despite the increasing activity in the area, the academic discussion on AI ethics has not reached the industry [5]. Through ECCOLA, we have attempted to make some of the contents of the IEEE EAD guidelines [4] and the EU Trustworthy AI guidelines [3] actionable, alongside other research in the area.

In developing ECCOLA, we have had three main goals for the method:

- To help create awareness of AI ethics and its importance,
- To make an adaptable, modular method suitable for a wide variety of SE contexts, and
- To make ECCOLA suitable for agile development, and to also make ethics a part of agile development in general.

In relation to the first goal, there is currently no way of benchmarking what is, so to say, sufficiently ethical in the context of AI ethics. This is arguably a limitation for any such method in the context currently. Benchmarking ethics is difficult and thus it is equally difficult for a method to have a proven effect in a quantitative manner. Moreover, ethical issues are often context-specific and require situational reflection. This has been why we have instead chosen to focus on raising awareness and highlighting issues rather than trying to provide direct answers for them. Raising awareness has also been a goal of the IEEE EAD initiative [4]. Raising awareness is important as the area of AI ethics is new for the industry.

ECCOLA provides a starting point for implementing ethics in AI. Based on our lessons learned thus far, we argue that ECCOLA facilitates the implementation of AI ethics in two confirmable ways. First, ECCOLA raises awareness of AI ethics. It makes its users aware of various ethical issues and facilitates ethical discussion within the team. Secondly, ECCOLA produces transparency of systems development. In utilizing the method, a project team produces documentation of their ethical decision-making by means of e.g. making notes on the note-making space in the cards and non-functional requirements in product backlog. Transparency is one key issue in AI systems, both in terms of systems and in terms of systems development [9]. These documents, as we have done while testing the method, can also be analyzed to understand how the method was used, aside from seeking to understand the reasoning behind the ethical decisions that were made.

The second goal has been based on the method-agnostic philosophy of the Essence Theory of Software Engineering [12]. Industry organizations use a wide variety of methods, from out-of-the-box ones to, more commonly, tailored in-house ones [19]. ECCOLA is not intended to replace any of these. Rather, ECCOLA is intended as a modular tool that

can be used in conjunction with any existing method. The use of ECCOLA in conjunction with agile methods and SE methods in general should still be further tested. For the time being, we received positive feedback relating to the modularity of ECCOLA when it was utilized in a project while using it in conjunction with SCRUM, an agile method (section IV C).

This, in turn, leads us to the third goal. As agile development is currently the trend, ECCOLA has been designed to be an iterative process from the get-go. However, during its iterative development, we noticed that a strict process was not a suitable approach due to being too heavy (section IV B). The users of the method opted out of adhering to the process and used the cards in a modular fashion despite the instructions. Now, ECCOLA is a modular tool by design. Being a card deck, this means that its users are able to select the cards they feel are relevant for each of their iterations, as opposed to having to go through the same process every time. Moreover, ECCOLA is intended to become a part of the agile development process in general. Ethics should not be merely an afterthought, but rather, a non-functional requirement, as well as a part of the user stories.

ECCOLA is a tool for developers and product owners. Ethics cannot be outsourced, nor can ethics be implemented by hiring an ethics expert [5]. AI ethics should be in the requirements, formulated in a manner also understood by the developers working on the system.

As governments and policy-makers have already begun to regulate AI systems in various ways (e.g. bans on facial recognition for surveillance purposes<sup>6</sup>), this trend is likely to only accelerate. With more and more regulations imposed on AI systems, organizations will need to tackle various AI ethics issues while developing their systems. This will consequently result in an increasing demand for methods in the area. While this will also inevitably result in the birth of various new methods, developed by companies, scholars, and standardization organizations alike, for the time being ECCOLA can serve as a starting point.

---

<sup>6</sup> <https://www.bbc.com/news/technology-51148501>

This is the author's version of the work. The definite version was published in: V. Vakkuri, K. -K. Kemell and P. Abrahamsson, "ECCOLA - a Method for Implementing Ethically Aligned AI Systems," 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 2020, pp. 195-204, <https://doi.org/10.1109/SEAA51224.2020.00043>

#### REFERENCES

- [1] C., Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nat Mach Intell* 1, 2019, pp. 206–215.
- [2] S., Pichai, AI at Google: our principles. Blog 2018. <https://www.blog.google/technology/ai/ai-principles/>
- [3] AI HLEG (High-Level Expert Group on Artificial Intelligence)," Ethics guidelines for trustworthy AI," 2019 <https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai>
- [4] The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems, "Ethically Aligned Design: A Vision for Prioritizing Human Well-being with Autonomous and Intelligent Systems", First Edition. IEEE. 2019. <https://standards.ieee.org/content/ieee-standards/en/industry-connections/ec/autonomous-systems.html>
- [5] V. Vakkuri, K.K. Kemell, J. Kultanen and P. Abrahamsson, "The Current State of Industrial Practice in Artificial Intelligence Ethics," in *IEEE Software*, vol. 37, no. 4, 2020, pp. 50-57.
- [6] A., McNamara, J., Smith, E., Murphy-Hill, "Does ACM's code of ethics change ethical decision making in software development?" *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering ESEC/FSE, 2018*, pp. 729-733.
- [7] J., Morley, L., Floridi, L., Kinsey, and A., Elhalal, "From What to How: An Initial Review of Publicly Available AI Ethics Tools, Methods and Research to Translate Principles into Practices," 2019, Preprint arXiv:1905.06876
- [8] A., Jobin, I., Marcello, and V., Effy, "The global landscape of AI ethics guidelines." *Nature Machine Intelligence* 1.9, 2019, pp. 389-399.
- [9] V., Dignum, "Responsible Autonomy," 2017, Preprint arXiv:1706.02513.
- [10] M., Ananny and K., Crawford, "Seeing without Knowing: Limitations of the Transparency Ideal and Its Application to Algorithmic Accountability," *New Media & Society* vol. 20(3), 2018, pp. 973–89.
- [11] M., Turilli and L., Floridi, "The ethics of information transparency" *Ethics and Information Technology*, vol. 11(2), 2009, pp. 105-112.
- [12] I., Jacobson, et al. "The essence of software engineering: the SEMAT kernel," *Communications of the ACM* 55.12, 2012 pp. 42-49.
- [13] G.,Theocharis, M., Kuhrmann, J., Münch and P. Diebold, "Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices," *Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015)*, volume 9459 of LNCS, 2015, pp. 149-167.
- [14] Pfeiffer, R.S. and Forsberg, R.P., "Ethics on the Job: Cases and Strategies" Wadsworth Publishing Company, California 1993.
- [15] V., Vakkuri, and K.K., Kemell, "Implementing AI Ethics in Practice: An Empirical Evaluation of the RESOLVEDD Strategy." *Software Business. ICSOB 2019. Lecture Notes in Business Information Processing*, Springer, Cham, vol 370 2019.
- [16] J., Leikas, R., Koivisto and N., Gotcheva, "Ethical framework for designing autonomous intelligent systems." *Journal of Open Innovation: Technology, Market, and Complexity* 5.1, 2019.
- [17] I., Jacobson, et al. "The Essentials of Modern Software Engineering," ACM, New York, 2019.
- [18] G., Susman, and R., Evered. "An assessment of the scientific merits of action research," *Administrative science quarterly* 1978 pp. 582-603.
- [19] H., Ghanbari, "Investigating the causal mechanisms underlying the customization of software development methods." *Jyväskylä studies in computing*, 258, 2017.



## IV

# **THE ENTREPRENEURIAL LOGIC OF STARTUP SOFTWARE DEVELOPMENT: A STUDY OF 40 SOFTWARE STARTUPS**

by

Anh Nguyen-Duc, Kai-Kristian Kemell & Pekka Abrahamsson, 2021

*Empirical Software Engineering*, vol. 26

DOI 10.1007/s10664-021-09987-z

Reproduced with kind permission by Springer.



Untitled Manuscript No.  
(will be inserted by the editor)

## **The entrepreneurial logic of startup software development: A study of 40 software startups**

Anh Nguyen-Duc<sup>1</sup> · Kai-Kristian Kemell<sup>2</sup> · Pekka Abrahamsson<sup>3</sup>

*Received: date / Accepted: date*

### **Abstract**

**Context:** Software startups are an essential source of innovation and software-intensive products. The need to understand product development in startups and to provide relevant support are highlighted in software research. While state-of-the-art literature reveals how startups develop their software, the reasons why they adopt these activities are underexplored.

**Objective:** This study investigates the tactics behind software engineering (SE) activities by analyzing key engineering events during startup journeys. We explore how entrepreneurial mindsets may be associated with SE knowledge areas and with each startup case.

**Method:** Our theoretical foundation is based on causation and effectuation models. We conducted semi-structured interviews with 40 software startups. We used two-round open coding and thematic analysis to describe and identify entrepreneurial software development patterns. Additionally, we calculated an effectuation index for each startup case.

**Results:** We identified 621 events merged into 32 codes of entrepreneurial logic in SE from the sample. We found a systemic occurrence of the logic in all areas of SE activities. Minimum Viable Product (MVP), Technical Debt (TD), and Customer Involvement (CI) tend to be associated with effectual logic, while testing activities at different levels are associated with causal logic. The effectuation index revealed that startups are either effectuation-driven or mixed-logics-driven.

**Conclusions:** Software startups fall into two types that differentiate between how traditional SE approaches may apply to them. Effectuation seems the most relevant and essential model for explaining and developing suitable SE practices for software startups.

**Keywords:** Software startup engineering, entrepreneurial logics, effectuation theory, case study, effectuation index, software engineering for startups

## **1. Introduction**

More and more software is developed by startup companies with limited resources and little operating history. Successful companies like Uber, Spotify, and Kahoot developed their software products during their startup stages. According to Pitchbook, investment in US startups only is more than 120 billion USD in 2019 (PitchBook, 2019). This substantial financial investment also implies a massive waste due to startups' high failure rate (Giardino et al., 2014). Previous research reveals critical challenges in both business and product development (Giardino et al., 2015). Consequently, attempts to deal with these challenges could eventually increase the odds of success, and the economic savings would be significant (Lindgren and Münch, 2016). The need to better understand software engineering (SE) in startups and provide relevant support for practitioners has been emphasized in the software startup research community (Unterkalmsteiner, 2016; Pantiuchina, 2017; Bajwa et al., 2017; Nguven-Duc et al., 2020). The emergence of software startup as a research theme is shown by an increasing number of studies on different engineering aspects in a startup context, for example, SE (Klotins, Unterkalmsteiner, Chatzipetrou, et al., 2019), requirements engineering (Melegati et al., 2019), software architecture (Fagerholm et al., 2017), software Minimum Viable Product (MVP) (Duc and Abrahamsson, 2016), and startup ecosystems (Tripathi et al., 2018). These studies explore the commonalities among startups regarding engineering processes, practices, and ways of working. We have better understood the demand for SE principles, processes, and practices in startup

---

<sup>1</sup> Business School, University of South Eastern Norway  
Gullbringvegen 36, 3800, Bø i Telemark  
Tel.: +47-48348496  
E-mail: angu@usn.no

<sup>2</sup> University of Jyväskylä, Finland

<sup>3</sup> University of Jyväskylä, Finland

companies, their challenges, and common ways of working. However, we do not understand why they adopt a particular workflow and under which circumstances they make these decisions.

State-of-the-art software startup research inherited from empirical SE research several preoccupations with normative studies on methods, methodologies, and models, and it lacks theories to understand and explain how and why things happen (Ralph, 2016). For instance, Aurum et al. (2003) adopted decision-making theories to understand the nature of requirement engineering activities. In response to this theoretical gap in software startup research, our previous work began to explore decision-making logics in software startups (Nguyen-Duc, Seppanen, and Abrahamsson, 2015; Kemell, Ventilä, Kettunen, and Mikkonen, 2019). Understanding the logic behind startup activities would enable the exploration of a systematic connection between decisions, activities, behaviors, and startup context, contributing to theory building in software startup research. Furthermore, patterns, or anti-patterns with their antecedent and consequent factors can be directly beneficial for startup companies.

Startups differ from established companies in the strong presence of entrepreneurial personalities, behaviors, decision-making, and leadership (Bygrave et al., 1991). Startups operate with a high level of uncertainty, multiple influences, and small team sizes, which magnify the influence of key persons, such as the CEO or CTO, on the project's success (Paternoster et al., 2014; Berg et al., 2018; Giardino et al., 2014). While entrepreneurial characteristics are evident in both information systems and business literature (Ojala 2015, 2016; Nambisan 2017), entrepreneurship rarely appears in SE research, either contextually or as a primary focus of the investigation. Tripathi et al. (2018) found that entrepreneurs' backgrounds influence how MVPs are developed. The following year, Melegati et al. (2019) found that startup founders strongly influence requirement engineering activities. However, neither study explores the logic underlying observed phenomena. Prescriptive methodologies have recently attracted considerable interest in entrepreneurship research (Sarasvathy and Dew 2005a, 2005b; Dew et al., 2009; Fisher, 2012; Berends et al., 2013; Reymen et al., 2015; Mansoori and Lackeus, 2019). There is a widespread research effort to identify the common logic or principles behind entrepreneurs' decisions and actions. A prominent example of an entrepreneurial logic is effectuation, presented as a set of heuristics any entrepreneur could use for business development in the context of high uncertainty (Sarasvathy and Dew 2005a, 2005b). The logic has been proposed in contrast to a traditional causation logic, in which entrepreneurs are plan-driven, perform their best within given constraints, and accept the possibility of a changed goal (Sarasvathy and Dew 2005a; Wiltbank et al., 2006; Read et al., 2009). As product development is critical for software startups, it is crucial to understand how entrepreneurial logic applies to software development activities.

In the quest to develop a theory of software startup engineering (Nguven-Duc et al., 2020), we want to understand further the logic behind decision-making (Boland 2008) in software startups. As a framework, we employ two entrepreneurial logic theories from entrepreneurship literature to investigate how requirement engineering, software design, construction, testing, and software development happen. To the best of our knowledge, this is one of very few attempts to incorporate entrepreneurial logic in the context of software development (Nguyen-Duc et al., 2017; Hevner and Malgonde, 2019). Of previously published studies, we are aware only of Khurum et al.'s (2015) use of the opportunity recognition theory and Hevner and Malgonde's (2019) assessment of effectuation theory in platform development. Unlike Hevner, we describe both effectual and causal logics in each SE activity. We also propose an explanatory model of the influences of entrepreneurial logic on software development activities in startups. This study aims to better understand the connections between the logic of startup founders and SE activities. Two research questions (RQs) were derived from the research objective:

RQ1: How do entrepreneurial logics apply to SE activities in startups?

RQ2: How do entrepreneurial logics apply to software product development at the company level?

The remainder of the paper is structured as follows: Section 2 contains background and related work, Section 3 explains the research method, Section 4 describes the results, Section 5 describes the findings, and Section 6 concludes the paper.

## 2. Related Work

The section presents important definitions used in this paper, background and related work about Software Startups, Software Engineering in Startups and Entrepreneurial logics. The key terminologies are summarized in Table 1.

Table 1 Key terminologies

Terms	Definitions	Reference
Software startup	Highly reactive and rapidly evolving software-intensive product development companies with an innovation focus and a lack of resources, working under uncertainty and time pressure	Section 2.1
Startup stage	Three main stages are pre-startup, startup and post-startup	Section 2.1.
Lead Users	Users who have a needs of general market but earlier than the crowd	Section 2.2
Minimum Viable Product	A version of a product with just enough features to be usable by early customers	Section 2.2
Entrepreneurial logic	The process of creatively defining, reframing and taking action to make sense out of business situations	Section 2.4
Sense making	A process by which people give meaning to their collective experiences	Section 2.4
Causal Logic	A process of pursuing a predetermined goal by acquiring needed resources, tools to achieve the goal	Section 2.4.1
Effectual Logic	A process of selecting among several possible goals with a pre-given set of resources	Section 2.4.2
Technical debt	Implied cost of additional rework caused by choosing a quick technical solution to meet an urgent demand instead of a sustainable approach that would take longer.	Section 2.5

## 2.1. Definitions of Software Startups

The term “startup” has been defined differently across various disciplines (Sutton, 2000; Ries, 2011; Blank, 2013; Unterkalmsteiner et al., 2016; Ghezzi, 2018; Steininger, 2019). Steve Blank (2013) describes a startup as a temporary organization that aims to create innovative high-tech products without a prior working history as a company. The author further highlights that the business and its product should be developed in parallel within the startup context. Eric Ries (2011) defines a startup as a human institution designed to create a unique product or service under extreme uncertainty. Rather than a formal company, a startup should be considered a temporary organizational state that seeks a validated and scalable business model (Unterkalmsteiner et al., 2016). A company with a dozen employees can still be in a startup state while it validates a business model or a market. As previous startup research has done (Berg et al., 2018), we define a startup as a highly reactive and rapidly evolving company with an innovation focus and a lack of resources, working under uncertainty and time pressure. We looked for companies that develop software products as their primary value proposition or include software as a significant part of their products or services. There are many different startup life-cycle models describing startups’ states of objectives, resources and business maturities. A startup model can have from three to seven stages, depending on the aspects they focus on. As adopted in our previous work (Nguyen-Duc et al. 2016, 2017), we define startups’ phases as the followings:

- Pre-startup stage: ideas are developed and need to be validated, startups in the quest for financial and human resources. Startup activities are carried out by founders or short-term hires. The purpose of this stage is to demonstrate business feasibility, team building and management. The common financing model is bootstrapping, family, friends and foes (FFF)
- Startup stage: prototypes are developed and experimented, startups have already figured out the problem/solution match. Some revenue is generated, but not necessarily over the break-even point. Founder seeks support mechanisms from startup ecosystems, learn to accelerate their business development. The common financing model is own funding and seed funding.
- Post-startup stage: products are extended, startups achieve the product/market match. Startups expand their customer bases, the revenue models are predictable and scalable. A hierarchical structure is formed within the startups. The common funding model is Series A, Series B, and other series

## 2.2. Agile development, User-centered Design and Lean startups

Contributions to agility and reactivity of product development are known from Agile (Beck et al., 2001), Lean (Gautam and Singh, 2008; Ries 2011), and User-centered Design (Norman 1986; Gothelf 2013) methodologies. Dealing with certain levels of uncertainties can be seen from different agile practices, such as short development cycles, collaborative decision-making, rapid feedback loops, and continuous integration enable software organizations to address change effectively (Highsmith and Cockburn, 2001; Beck and Andres, 2004). In startup contexts, Giardino et al. showed that agile practices are adopted, but in an ad-hoc manner (Giardino et

al., 2014). Pantiuchina et al. studied 1256 startup companies and reported that different agile practices are used to different extents, depending on the focus of the practices (Pantiuchina et al. 2017). The authors found that speed-related agile practices are used to a greater extent in comparison to quality-related practices. Recently Cico et al. reported that startups in their growth phases do apply Agile practices in various ways. Strict adoption of agile methodology seems not to be perceived critically, and in some situations, it is difficult to apply agile practices due to the nature of developing products (Cico et al., 2020).

Lean startups with the focus on forming hypotheses about businesses, building experiments to evaluate them (Ries 2011), had a large impact on startup and research communities. Minimum Viable Product (MVP) is a central concept of the approach, defined as a version of a product with just enough features to be usable by early customers (Ries 2011). Bosh et al. discussed why few practitioners apply Lean Startup methods because of the lack of guidelines for method operationalization (Bosch et al., 2013). Other factors influencing the implementation of Lean Startup are also reported, such as the costs of prototyping in particular (Ladd et al., 2015), experience and knowledge about the methodology (Nguyen-Duc et al., 2016, 2017), and experimentation in general (Gemmell et al., 2012).

Customer Development is another popular paradigm that focuses on customers upfront, i.e. developing the customers rather than products in the early stages of startups (Blank 2007; Blank & Dorf, 2012; Alvarez 2014). So startups are advised to search for the right customers to test their business hypotheses and thus obtaining validation or refutation of the overall business model. This relates to the marketing practices of lead users who (1) face the needs that will be general in the market, but face them much earlier than the crowd, (2) are positioned to benefit significantly by obtaining the solution to those needs (von Hippel, 1986). User-Centered Design is also a relevant paradigm for certain types of startups, as they aim for creativity and empathy for designing user-centric solutions and helping developers to change their mindset on how to approach a problem and envision its solution (Signoretti et al., 2019). Hokkanen et al. studied User Experience (UX) practices in startups and suggested that startup products need to fulfill minimal functional and user experience requirements (Hokkanen et al., 2015).

Startups, in general, do not follow one or many of these methodologies strictly. This applies not only to startup companies, as a recent large-scale survey in European software companies showed that modern software and system development does not follow any blueprint and adopt different hybrid approaches (Tell et al, 2017). The understanding of which compositions of development methods, i.e. Agile, Lean, etc that actually work in software development contexts is missing (Tell et al, 2017). In this work, we aim to understand the possible links between adopted development practices with the entrepreneurial logics.

### 2.3. Software Engineering Models for Startups

The need for understanding and modeling SE phenomenon in startup companies has been recognized in SE literature. Giardino et al. (2016) explained a phenomenon of accumulated TD in startup contexts when product quality is a low priority and the startup team is more focused on speeding up development. The authors pointed out that lack of resources is the main driver for the observed product development patterns; however, they did not explain how the limited resource leads to the lack of focus on quality. Nguyen-Duc et al. (2016) described startup development patterns by looking at the co-evolution of product and business as an inter-twined process: startups need entrepreneurial skills and project management skills when hunting implementing opportunities (Nguyen-Duc et al., 2015). These models take into account the influence of business factors in decisions on product development. However, the number of cases investigated at that time was limited. Fagerholm et al. (2017) implemented the Build-Measure-Learn cycles (Ries, 2011) as continuous experimentation systems, where the new product idea can be hypothesized and tested.

Some studies explored particular activities of product development, such as MVP development or requirement engineering. Nguyen-Duc et al. (2017) described how MVPs are used in different software startups, and Tripathi et al. (2018) revealed how the supporting roles of startup ecosystem elements influence MVP development. More recently, Melagati et al. (2019) presented how founders and other factors influence startups' requirement engineering activities. These studies acknowledged the impact of entrepreneurs on SE activities; however, they do not have a theoretical model to explain this impact. These studies call for the adoption of decision-making theories to fill the gap. More recently, Klotins, Unterkalmsteiner, Chatzipetrou, et al. (2019) looked at commonalities among startups' goals, challenges, and practices. The authors showed that startups share the same SE challenges and practices with established companies; however, startups need to evolve multiple activities simultaneously. The study described product development startups from a project management perspective to consider planning, measuring, and controlling activities. This assumption suggests a plan-driven logic when

looking at the startup development process and might lead to a similar observation when comparing these plan-based activities to those of established companies. In contrast to a plan-driven and controlled approach, effectual logic adopts means-driven, emergent, and flexible mechanisms to deal with the environment's uncertainty. Previous studies in SE have suggested that the availability of resources can influence the occurrence of engineering phenomena, i.e., TD (Giardino et al., 2016) and the choice of which MVP to implement (Nguyen-Duc, Dahle, et al., 2017).

#### 2.4. Effectuation and Causation Logics in Entrepreneurial Decision Making

Entrepreneurial logic is defined as a process of creatively defining, reframing and taking action to make sense out of situations that require new assumptions and understandings (Cunhingham et al. 2002). Sensemaking is defined as "the ongoing retrospective development of plausible images that rationalize what people are doing" (Weick 1995, Weick et al. 2005). In the purpose of making sense from startup situations, two kinds of entrepreneurial logic that have recently gained research attention are the logics of effectuation and causation (Sarasvathy, 2001; Alvarez and Barney, 2005; Fisher, 2012; Reymen, 2015). Below, we present their definitions and examples in the context of software development.

##### 2.4.1. Causal Logic

In a nutshell, causal logic describes a process of pursuing a predetermined goal by acquiring needed resources, tools to achieve the goal. The causal logic focuses on the predictable aspects of an uncertain future and follows the logic of "to the extent we can predict the future, we can control it" (Sarasvathy, 2001, p. 7). An example of this approach is to conduct a project in a large company. When a project manager is assigned to the project, he perhaps needs to gather his team to apply for extra resources if needed. He needs to be aware of project constraints and perform to achieve the predetermined goals of the project. The project manager's attitude towards unexpected contingency is avoided. He relies on accurate predictions, careful planning, and focusing on predetermined objectives. In the causation model, startups focus on competition and constrain task relationships with customers and suppliers. For instance, the project manager needs to manage the relationships with external stakeholders to limit their possible negative influences (delays in the project schedule, unexpected costs, and other unanticipated problems). The causation model highlights the action to maximize returns by selecting optimal approaches (Sarasvathy and Dew, 2005b). The manager will prioritize analytical calculations and pursue an optimized approach.

##### 2.4.2. Effectual Logic

An effectual logic describes a process of selecting among several possible goals with a pre-given set of resources (Sarasvathy, 2001; Barney, 1991). The effectual logic focuses on the controllable aspects of an unpredictable future and follows the logic of "to the extent we can control the future, we do not need to predict it" (Sarasvathy, 2001). For example, a startup that is a spin-off from a university has technological patents. The startup decides to develop different business models leveraging the application of the patents. The effectuation-driven startup tends to involve as many people as possible in the early stages to generate value for the startup. Instead of focusing on maximized returns, the effectuation-driven startup examines how much one is willing to lose on a startup journey. In our example, the startup team needs to calculate and commit only the resources, time, and effort that they can tolerate wasting.

#### 2.5. The Need for Entrepreneurial Logic in Software Development

Traditional software development approaches start with a particular goal and realize it through a linear or iterative development process, which is largely overlap with causal logic. Significant parts of SE research base on a prescriptive assumption that software development projects can be guided by reference frameworks, processes, techniques, and tools. When managing a software project, one could assume a certain level of control based on plan-driven and systematic working manners (Klotins, Unterkalmsteiner, Chatzipetrou, et al., 2019) where project context, such as market, customers, and other ecosystem elements, are somewhat identified as a priori.

Table 2 Software startup phenomenon and their possible connections to entrepreneurial logics

Phenomenon	Description	Reference	Judgment
Software pivot	A pivot is a strategic change designed to test a fundamental hypothesis about a product, business model, or growth engine.	Model of pivot triggering factors (Bajwa et al., 2017; Bajwa,	A certain type of product pivot would be desirable and plan-driven. But most of the pivots are triggered by external factors and reflect the effectual logic

		2020; Khanna et al., 2018)	
Technical debt (TD)	implied cost of additional rework caused by choosing a quick technical solution to meet an urgent demand instead of a sustainable approach that would take longer.	Greenfield model of software startups (Giardino, 2016; Seaman and Guo, 2011)	Startups accumulate TD, but its nature might be different from large companies in that TD is a way to manage tolerable loss in effectual logic.
Minimum Viable Product (MVP)	A version of the product to collect validated learning	MVP-based learning (Nguyen-Duc et al., 2017; Duc and Abrahamsson, 2016)	Startups develop many MVPs but in order to gather necessary learning, they need a more plan-driven approach
Customer involvement in product development	Customers involve early and often in requirement, design, and testing activities	Continuous involvement (Nguyen-Duc et al., 2017; Melegati et al., 2019; Yaman et al., 2016)	Effectuation-driven companies encourage the contribution of external stakeholders in co-creating company value

Software development in startups often needs to deal with multiple-influenced and rapidly changing business and working environments, which makes effectual logic relevant (Giardino et al., 2014; Giardino et al., 2016; Bajwa et al., 2017). In software startups, product development is often essential for the success or failure of the company. They are often limited in resources and work under pressure to prove their products or services to attract funding. Such settings make formal software development paradigms less applicable (Pantiuchina et al., 2017; Kemell et al., 2019). Notably, previous studies also reported that startups’ working way is contingent on their environment (Nguyen-Duc et al., 2015; Nguyen-Duc et al., 2016; Kemell et al., 2019). Effectual logic could help explain decisions or activities taken in resource-constraint situations. Brettel et al. showed that effectuation is positively linked to process output and efficiency in highly innovative RandD projects (Brettel et al., 2012). Similarly, these logics could be relevant to SE activities in startups. Several previously studied technical concepts in software startups, such as MVP and TD, can be explored further under entrepreneurial logic (as shown in Table 2).

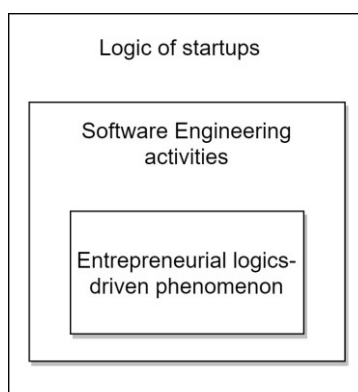


Figure 1 A conceptual framework of entrepreneurial logics in software startups

In this study, we argue that entrepreneurial logic can help to understand specific patterns between entrepreneurial contexts and how product development activities are chosen. Mansoori proposed a three-tiered framework for the mapping of entrepreneurial processes onto the three levels: logic, model, and tactics (Mansoori, 2015, 2020). At the logic level, principles for startups include the notion of uncertainty (epistemological or ontological), view of the future (predictable or completely unknown), nature of the process (discovery or creation), epistemological discussions (realism or constructivism), and relation to external stakeholders (transactional or generative) (Mansoori 2015). At the model level, there are often organized sequences of operations and interactions for guiding entrepreneurial actions. At the tactic level, there are activities, exercises or practices that are in line with the underlying logic and the prescribed model. While the original framework applies to entrepreneurial activities in general, we adopted it to the software engineering model and activities, as seen in Figure 1. At the model level,

we relate the entrepreneurial logic to SE processes, i.e. requirement engineering, software design, implementation, and testing. At the tactic level, we will extract specific SE activities and practices that characterize entrepreneurial logic. This study takes the first step towards entrepreneurial software development by exploring the connection between entrepreneurial logic and software product development activities. Our intention is not to predict startups' behaviors or to classify them as either kind of logic.

### 3. Research Methodology

Our research goal was to generate new knowledge about the logic behind software product development in startups, which needs to be investigated in its natural setting. Inductive research with a bottom-up exploration of evidence and conclusions generated from this evidence is a suitable approach commonly adopted in empirical SE research (Seaman, 1999; Wohlin and Aurum, 2015; Ayala et al., 2018; Khurum et al., 2015). All the different research methodologies have their place in software engineering, and each approach has value for the software engineering practitioner (Easterbrook et al., 2008). The possible choices for such an empirical study were exploratory, descriptive, explanatory, and evaluation research (Collis and Hussey, 2009). Compared to other SE research fields, software startup research is still a growing field with a limited understanding of engineering-specific activities in this context (Unterkalmsteiner et al., 2016; Berg et al., 2018; Klotins et al., 2019). From philosophical perspectives, the study adopted a mixed view between interpretivism and positivism. On one hand, the study has many assumptions of interpretivism, i.e. research must be interpreted within the context in which it takes place, and research findings are subjective (Walsham, 1995). Besides, the goal of this research is to provide deep insight regarding entrepreneurial scenarios, not to confirm or test a hypothesis. Therefore, we endeavored to explore software development from entrepreneurial perspectives descriptively; we explain how to plan and organize startups' work accordingly. Since these phenomena involve mainly human factors, it is vital to evaluate human perceptions of the subject (Easterbrook, 2008). On the other hand, we adopt the concept of cognition from positivism, emphasizing the role of empirical evidence in the formation of ideas, rather than innate ideas or traditions. Systematic approaches to collect and analyze evidence is pursued towards reproducible findings and logic-based science. By collecting data in the form of responses to standardized questions, i.e. survey research, accumulated evidence can constitute facts.

It is not uncommon in SE/Information Systems (IS) research for empirical studies adopting both paradigms (Runeson and Höst, 2009, Stol et al., 2016). In a mixed-research approach, qualitative data can be coded quantitatively" by counting words and categorizing statements (Trochim, 2001) or combinations of survey data with case studies (Ralph, 2015). We have adopted the approaches in our previous work (de O. Melo et al., 2013; Ayala, 2018). To gather and interpret the evidence needed to answer our research questions, we conducted semi-structured interviews with startup cases. Depending on the in-depth knowledge of a case, qualitative research can focus narrowly on a few case studies, or tackle a broader scope. We used the same set of key questions repeatedly in a relatively large number of cases (N=40). We aim at observing both frequency distribution and systematic and thematic patterns across interview cases.

This study could have been carried out at the activity, team, project, and company levels. To associate entrepreneurial logic with SE, we needed to look at specific activities and their context; hence, the first analysis was performed at the activity level (RQ1). Since we collected data from different companies, it was straightforward to then perform the second analysis at the company level: in other words, a cross-case analysis.

#### 3.1. Case Selection

The challenge of identifying proper startup cases and differentiating the similar phenomena represented among them — freelancers, SMEs, or part-time startups — is well known in software startup research (Unterkalmsteiner, 2016; Berg et al., 2018). Based on the successful approaches adopted in previous studies (Klotins, Unterkalmsteiner, Chatzipetrou, et al., 2019; Berg et al., 2020), we defined five criteria for our case selection:

- A startup that has at least two full-time members, so their MVP development is not individual activities
- A startup that operates for at least six months, so their experience can be relevant
- A startup that has at least a first running prototype, so the prototyping practice is a relevant topic
- A startup that has at least an initial customer set, i.e., first customer payments or a group of users, so that certain milestones in the startup's process are made
- A startup with software as the central part of its business core value

We intended to conduct multiple interviews in each startup to achieve data triangulation (Boyatzis, 1998); however, most startups could only provide a single interview. We obtained multiple follow-up interviews in seven

cases (S01–S05, S07, S08), which provide the main insights. The other 33 cases, with a single interview, extend, and confirm the findings from the principal cases. All the information about the cases was collected via internet research and written documents provided by the companies to address gaps left by the lack of follow-up interviews. The characteristics of the cases studied are summarized in Section 3.3.

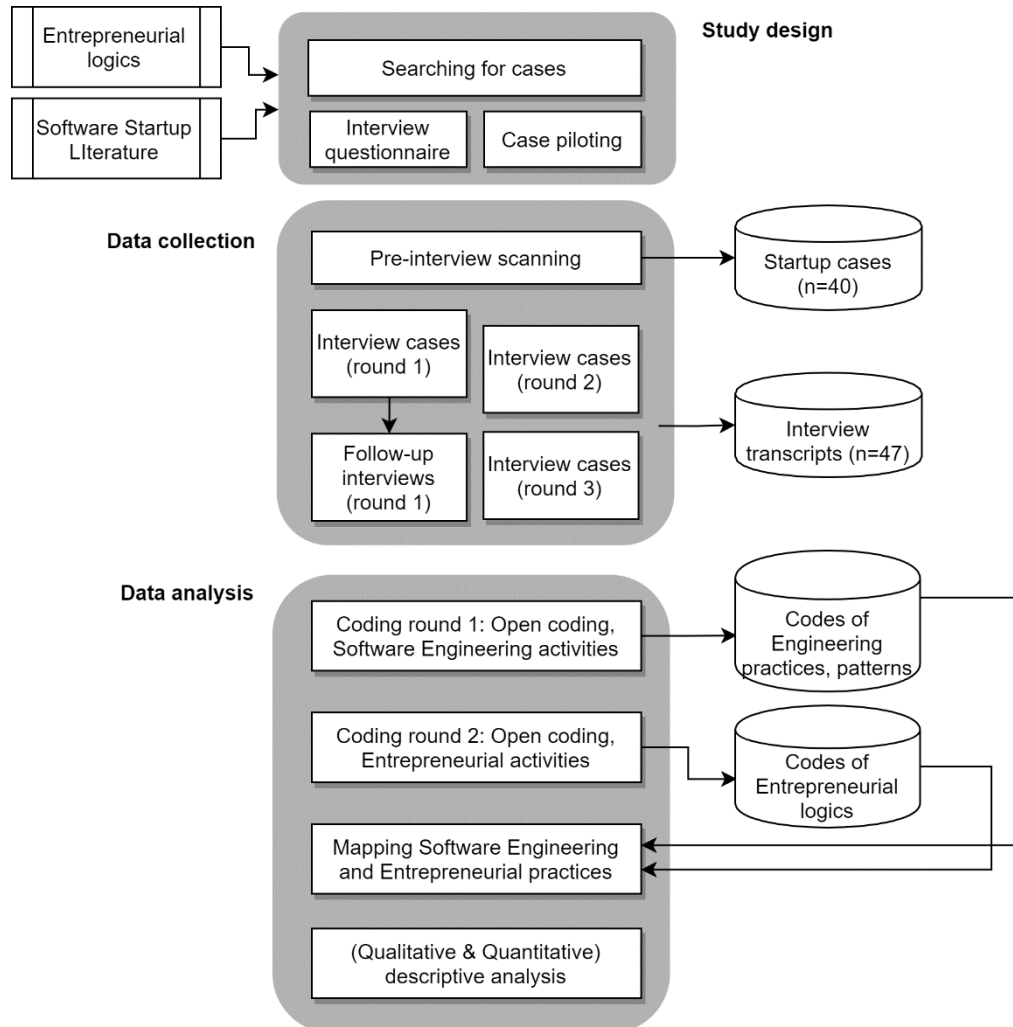


Figure 2 Data collection and analysis process

### 3.2. Data Collection

The main data collection methods were semi-structured interviews, participant observations, face-to-face discussions with project leaders, and document analysis. The identification and collection of data were performed in three rounds. The first round was conducted from March 2015 to February 2016; data collection was mainly done by the first author and a research associate. The second round was conducted during September 2016 and January 2017. The third round was conducted from September 2017 to June 2018; this data collection was performed collectively by the first author and graduate students at the Norwegian University of Science and Technology. A consistent approach was undertaken to collect data (as shown in Figure 2). The data collection process was as follows:

- Step 1: Identifying cases. Contacts for startups were searched via four channels: (1) startups within the authors' professional networks; (2) startups in the same towns as the authors and from Startup Norway; and (3) startups listed in the Crunchbase database. We also included contacts we made at startup events, such as the Norwegian Investment Forum, Startup Weekend, and Hackathons.
- Step 2: Feasible analysis. We spoke with software startups in coworking spaces and incubators in Trondheim, Norway, to become familiar with startup scenes and their current issues.



- Step 3: Study design. Several more interviews were conducted both face-to-face and remotely to build data collection equipment. The interview guideline was modified from an existing one, which focused on the topic of startup pivots.
- Step 4: Case piloting. Case analysis was conducted using information available from the internet or provided by the case companies that allowed a holistic understanding of each case and provided more substantial evidence for the conclusions drawn from the interviews. This step was conducted before proceeding to the actual interview with startups.
- Steps 5 and 6: Data collection. Interviews allowed us to collect information in the participants' own words rather than by limiting them to predefined response choices on a survey (Oates, 2005). We chose to conduct semi-structured interviews, as these are expected to give a researcher the flexibility to probe deeper into unforeseen information that may emerge during interviews (Seaman, 1999). Each interview lasted between 40 and 70 minutes. The number of interviews in each round is shown in Table 3.

Table 3 Data collection rounds

Round	No. of contacts	No. of cases	No. of interviews
1	219	20	27
2	40	7	7
3	47	13	13

The final contact list included 306 startups from the USA, Norway, Sweden, Finland, Italy, Germany, Spain, the Netherlands, Singapore, India, China, and Vietnam. We approached the companies on the final list to search for participants; many startups expressed their interest in the study results but did not have time to participate. These companies responded to our call for participation with sentiments similar to “[t]he research appears interesting and relevant to our experience. Unfortunately, we do not have the resources and time to participate in such a survey.” Besides some large companies who were not interested in the research, we did not see the difference between the ones who accepted and the ones who refused to participate. By emailing and talking via phone, professional networks and nearby startups have a slightly lower turn over rate than startups from Crunchbase. However when approaching startups via personal introduction or meet in person, there is significantly higher chance to acquire their participation.

Excluding startups that were not interested in the research or startups that did not meet our selection criteria, the final number of eligible cases was 40 startups (turnover rate ca. 13%). Among them 25% of the total number of cases come from our convenient networks, 70% of the cases are systematically selected and collected from physical interviews, 5% of the cases are from the CrunchBase database. Some startups required that the authors sign a non-disclosure agreement with the companies; this step was essential to establish a formal link between the researchers and the participating startups and ensure the data confidentiality the companies required to feel more comfortable with our observations of their internal activities.

Table 4: (Common parts of ) the interview guidelines

<p>Section 1: Business background</p> <ul style="list-style-type: none"> <li>Please tell us about your product and your company</li> <li>How was the current software product developed ?</li> <li>What is your team competence? How is it evolved over time?</li> <li>What is your current market?</li> <li>What is your business model?</li> </ul> <p>Section 2: Idea visualization and prototyping</p> <ul style="list-style-type: none"> <li>Could you tell us about the time when: (1) the first idea came to your mind, (2) the first prototype completed, (3) the first payment customer</li> <li>How can you achieve the problem/solution fit with your prototype?</li> </ul> <p>Section 3: Product development</p> <ul style="list-style-type: none"> <li>How many times have you changed? About the most significant pivot: How decisions are made?How was the current software product designed? What is the most challenging issue?</li> <li>How was the current software product implemented and tested? What is the most challenging issue?</li> <li>How was the current software product maintained and extended? What is the most challenging issue?</li> </ul>
---

In the first round of data collection, most participants answered a simple pre-interview questionnaire in which they filled out basic information about themselves and the company. In some cases, accessing sprint planning documents, product specifications, pitching slides, and communication mailing lists extended our knowledge about startup product development activities. Participant observation occurred in cases S02 and S03, where the first author involved in these cases was either a consultant or a co-founder. These measures facilitated more efficient interviews, as the first author possessed more knowledge about the case and could use less time on formalities. Most of the interviews were conducted by the first author. The author also took notes to mark essential concepts that came up in the interviews. Later on, all the interviews were transcribed using a freelancing service. A researcher in our network recommended the service, and the pilot test of the service was conducted before the study adopted it. The total number of transcripts was 313 A4 pages. In the second and third rounds of data collection, the first authors attended some interviews. Most of the interviews in these rounds were conducted by either graduate students or associated researchers. Although interview questions were slightly different among the three rounds of interviews, the interview structure and key questions remained the same. The interviewees were typically asked about (1) the business background, (2) idea visualization and prototyping, and (3) product development. The common key questions is described in Table 4.

### 3.3. Case Demographics

As shown in Figure 3, our cases vary significantly in terms of application domains. The investigated companies deliver software platforms in healthcare, information technology infrastructure, education, logistics, sales, and marketing. Investigated MVPs included software-intensive products (e.g., mobile apps, dynamic webs, and data analytics) and hardware-relevant products (e.g., Internet of Things platforms). The startup cases present a large spectrum of market segments: prominent startups (65%) targeted a niche market, such as hyper-local news readers, a population of high school pupils and college students, IoT product developers, software developers, and sale-intensive organizations; 35% of the cases currently follow a business-to-business (B2B) model; and the rest operate a business-to-customer (B2C) model. From a geographical perspective, the case sample is biased toward startups serving the Nordic and UK markets: these constitute 75% of the study's total cases. Other geographical markets include the USA, Germany, France, the Netherlands, Poland, Singapore, Hong Kong, and Vietnam. In terms of their headquarters' locations, the demographic representation of the case is shown in Figure 4.

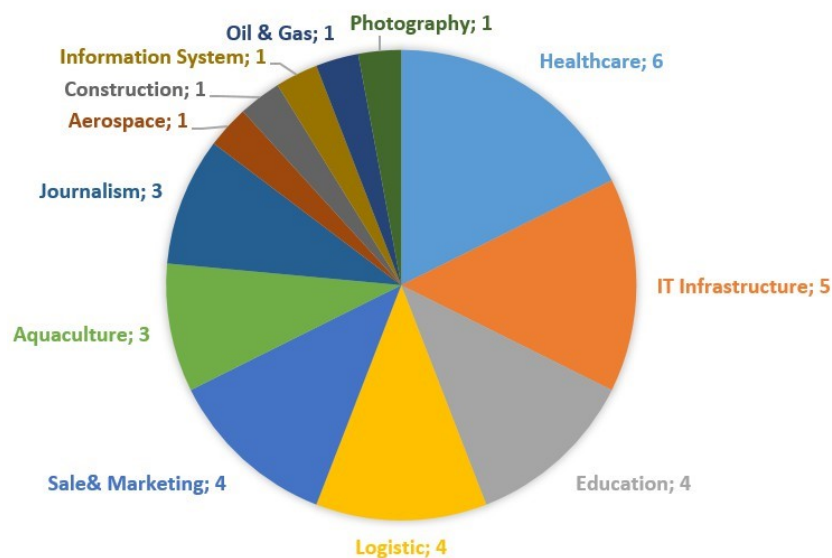


Figure 3 Distribution of startups in application domains

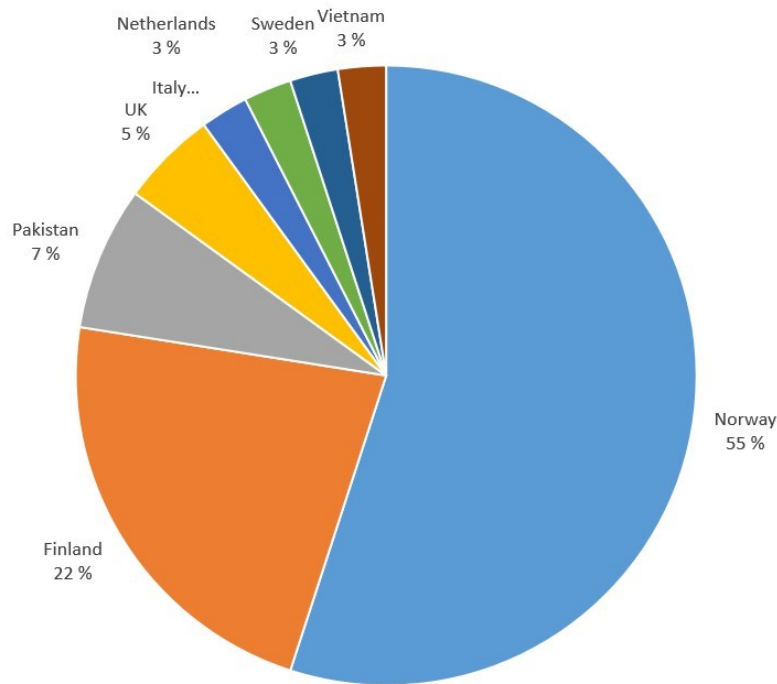


Figure 4 Distribution of startups in terms of headquarters' locations

Team sizes varied from 2 to 85 people, but most of the study's startups (27 out of 40) had a team of between 3 and 20 people. These headcounts include full-time workers employed during the study period, regardless of whether they were included on the payroll at that time. For example, one startup consists of the CEO, CTO, a designer in Norway, and an outsourcing team of six full-time developers in India. The portion of engineers in startups in our sample ranges from 33% to 100%. In many startups, the team is collocated and unstructured. In other cases, it is typical to observe organizational structures with a separation between product development teams and sales teams. Most of our startups (85% of the total number of cases) are financial bootstraps: they fund the development of products and services through internal cash flow and are cautious with their expenses. Most of the bootstrap startups studied received financial support from their governments, incubators, and accelerators, and startup programs. Some cases were initiated by an investor who secured a stable income for the team. Some cases were in the post-startup phase, with annual revenue above EUR 1 million. Some other startups had invested more than EUR 1 million. The financial information for the remaining startups is either unknown or indicates that they were struggling with their cash flow at the time of the study. The detail profiles of our cases as well as their financial situations are reported in Table 5.

Table 5: The profiles of the startups included in the study

Cases	Country	Product	Application domain	Years Operational*	Current Stage	No. People	Annual Revenue	Source	Startup type
S01	Italy	Photo trading platform	Arts, Entertainment and Recreation	4	Pre-Startup	15	<50k Eur	Source1	1
S02	Norway	Hyper-local news platform	Professional, Scientific, and Technical Services	1	Pre-Startup	2	110k Eur	Source1	2
S03	Norway	Shared shipping platform	Transportation and Warehousing	3	Startup	6	<10k Eur	Source2	1
S04	Norway	Digitalized construction management process	Construction	5	Post-Startup	9	>300k Eur	Source2	1

S05	Finland	Underwater camera product	Agriculture, Forestry, Fishing and Hunting	4	Pre-startup	3	Unknown	Source2	1
S06	Norway	Sales visualization	Professional, Scientific, and Technical Services	3	Post-startup	18	>1.5 mil. Eur	Source2	2
S07	Vietnam	Shop location app	Professional, Scientific, and Technical Services	4	Startup	14	~200k Eur	Source1	1
S08	Norway	Event and ticket platform	Professional, Scientific, and Technical Services	4	Startup	4	Unknown	Source2	1
S09	UK	Game-based classroom learning tool	Education	9	Post-startup	12	~2 mil. Eur	Source1	2
S10	Norway	IoT OS platform	Professional, Scientific, and Technical Services	4	Startup	3	>150k Eur	Source2	1
S11	Norway	Ticketing system	Professional, Scientific, and Technical Services	4	Startup	5	>150k Eur	Source2	1
S12	Norway	eLearning	Education	8	Startup	3	Unknown	Source2	1
S13	UK	Shipping services	Professional, Scientific, and Technical Services	2	Startup	3	Unknown	Source3	2
S14	Sweden	Journalism publishing	Professional, Scientific, and Technical Services	11	Startup	16	Unknown	Source3	2
S15	Norway	Secondhand marketplace	Professional, Scientific, and Technical Services	6	Pre-startup	2	Unknown	Source2	1
S16	Norway	Smart grid	Utilities	5	Startup	30	Unknown	Source2	1
S17	Norway	Simulation-based training	Education	7	Startup	7	Unknown	Source2	2
S18	Holland	Software development services	Professional, Scientific, and Technical Services	7	Startup	5	Unknown	Source3	1
S19	Norway	Mobile alert services	Professional, Scientific, and Technical Services	9	Startup	5	Unknown	Source1	1
S20	Norway	eLearning	Education	14	Startup	13	350k Eur	Source2	2
S21	Norway	Fish farm tracking system	Agriculture, Forestry, Fishing and Hunting	1	Pre-startup	6	Unknown	Source2	1
S22	Norway	Networks of connected camera	Professional, Scientific, and Technical Services	1	Startup	10	Unknown	Source2	1

S23	Finland	Underwater drone	Agriculture, Forestry, Fishing and Hunting	4	Pre-startup	4	Unknown	Source2	1
S24	Finland	Tracking devices for shipment	Professional, Scientific, and Technical Services	2	Post-startup	85	>8 mil. Eur	Source2	2
S25	Finland	Muscle operation measure	Health Care and Social Assistance	2	Pre-Startup	20	100k+ Eur	Source2	2
S26	Pakistan	Smart home solution	Manufacturing	2	Pre-startup	8	Unknown	Source1	1
S27	Pakistan	Smart wheelchair	Health Care and Social Assistance	1	Pre-Startup	3	Unknown	Source1	1
S28	Finland	Connecting healthcare services to home	Health Care and Social Assistance	5	Startup	5	Unknown	Source1	1
S29	Pakistan	Smart home devices	Professional, Scientific, and Technical Services	1	Pre-Startup	5	Unknown	Source1	1
S30	Finland	UI framework for mobiles	Professional, Scientific, and Technical Services	4	Pre-Startup	3	Unknown	Source1	1
S31	Finland	Aeronautical engineering services	Manufacturing	5	Startup	Unknown	Unknown	Source2	1
S32	Norway	IoT solution for gas supplier	Utilities	2	Startup	8	Unknown	Source2	1
S33	Norway	Personal hydration monitoring device	Health Care and Social Assistance	2	Startup	10	Unknown	Source2	2
S34	Finland	Enterprise information management solution	Professional, Scientific, and Technical Services	5	Startup	10	Unknown	Source2	1
S35	Norway	Ear device	Health Care and Social Assistance	2	Startup	5	Unknown	Source2	1
S36	Finland	Wireless earplug with active noise cancelling	Health Care and Social Assistance	2	Pre-startup	10	2000 orders/year	Source2	2
S37	Norway	Autonomous drones	Transportation and Warehousing	3	Pre-startup	7	Unknown	Source2	2
S38	Norway	Sensor-based detecting systems	Health Care and Social Assistance	2	Startup	7	Unknown	Source2	1
S39	Norway	Security for IoT	Professional, Scientific, and Technical Services	2	Startup	3	Unknown	Source2	1
S40	Norway	Drone control glove	Professional, Scientific, and Technical Services	1	Pre-startup	13	Unknown	Source2	2

Notation: Source1: startups within the authors' professional networks; Source2: startups in the same towns as the authors and from Startup Norway; Source3: startups listed in the Crunchbase database

### 3.4. Data Analysis

Data analysis included three steps: (1) labeling SE activities, (2) identifying entrepreneurial logics that occurred in each case, and (3) mapping the entrepreneurial logics and SE activities.

#### 3.4.1. Labelling SE Activities

We applied a thematic analysis, which is commonly seen in empirical SE research (Cruzes and Dyba, 2011). The objective of our thematic synthesis process was to answer the research questions and come up with a model of higher-order themes describing a way of software development in startups. Braun et al. (2006) suggest six steps for a thematic analysis: (1) familiarizing with data, (2) generating initial codes, (3) searching for themes, (4) reviewing themes, (5) defining and naming themes, and (6) producing the report. As suggested in the literature, we adopted open coding (Braun et al., 2006; Wohlin and Aurum, 2015). Sentences that mentioned SE activities and their contexts are labeled. We developed a taxonomy of startup knowledge and practice areas, including SE knowledge areas from SWEBOK (Bourque and Fairley, 2014). We tried to produce as many codes as possible to avoid missing any relevant or interesting information. The coding scheme for SE activities includes:

- P0. SE (general)
- P1. Requirement Engineering
- P2. Product Design
- P3. Software Construction
- P4. Software Testing
- P5. Software Maintenance
- P6. Software Process Management

#### 3.4.2. Identifying Effectuation-driven and Causation-driven Behaviors

To understand the logic behind decisions in a startup, we need first to understand the startup journey and important milestones. The first step was to read through the transcribed interviews to generate initial ideas and identify possible trends or patterns. For each case, we extracted texts related to critical events that occurred during the startups' journeys. Our approach is similar to previous studies that used key events identified through information from the interviews (de O. Melo et al., 2013; Reymen et al., 2015; Reymen et al., 2017; Fagerholm et al. 2017). Key events were defined as actions or decisions taken by the entrepreneurial teams to create the venture (Reymen et al., 2017). Examples of these events were introducing the first product idea, acquiring funding, initiating collaboration with a supplier, developing the first MVP, product demonstration and launch, hiring employees, and significant pivoting (Reymen et al., 2015). Such events were collected from critical people (e.g., CEOs or CTOs in the startups) and reflected their intentions. The decisions made by external stakeholders were placed in the context category. For each case, we tried to capture each event's timestamp—the pre-startup, startup, and post-startup phases—to plot each case's story in chronological order.

We attended to describe the startup event from participants' perspectives or views. We look for the meaning behind startups' events and activities. Daher stated that “the study of meaning does not directly refer to actual experience, but to the way the self considers its past experience” (Daher et al., 2017). And the meaning of being effectuation-driven or causation-driven is reflected from what the interviewees' own wills.

Identifying entrepreneurial logic was crucial to categorizing a case as either effectuation or causation. As in previous work, we coded entrepreneurial logic at the company level: we created a balanced coding scheme consisting of two theoretical categories based on effectuation and causation theory, i.e., one effectuation and one causation category with four dimensions for each category (Chandler et al., 2011; Reymen et al., 2015). We reused a set of empirical indicators (Sarasvathy, 2001; Read et al., 2009; McKelvie et al., 2020) and modified the coding scheme to make it relevant to the SE context. The coding scheme for the effectual logic includes:

##### **E1 Basis for acting: means-oriented**

- E1.1 Building product mainly on an internal knowledge base and external existing owned resources
- E1.2 Defining a general product development plan without concrete details
- E1.3 Using internal or resource and infrastructure in the local environment
- E1.4 Decisions mainly based on personal preferences
- E1.5 Opportunities, ideas, and requirements come from existing contacts

##### **E2 Attitude towards unexpected events: leverage**

- E2.1 Accepting and incorporating unexpected changes, ready for pivots
- E2.2 Changing and adapting any potential plans made to accommodate unforeseen events
- E2.3 Actively exposing to external stakeholders with an open mind

E2.4 Positively reacting to and incorporating unforeseen developments

**E3 Attitude towards outsiders: partnerships**

E3.1 Reaching trust-based flexible stakeholder agreements and commitments

E3.2 Co-create business with stakeholders

E3.3 Engaging in stakeholder collaborations to pursue opportunities

E3.4 Exposing MVPs to potential clients early on

**E4 View of risk and resources: affordable loss**

E4.1 Be willing to make affordable personal sacrifices (including nonmonetary)

E4.2 Finding unused resources in a local environment (including subsidies)

E4.3 Investing limited, small amounts of personal money, time, and effort

E4.4 Managing growth expectations and ambitions

E4.5 Limiting stakeholders' commitments to levels that are uncritical to them

The coding scheme for the causal logic includes:

**C1 Basis for acting: goal-oriented**

C1.1 Base actions upon expectations (market, technology, policy trends) and predictions (of founders, board members, investors)

C1.2 Defining and pursuing project goals, product, customer needs, or market goals (more specific than "profit" or "a better planet")

C1.3 Defining and satisfying organizational needs (personnel, organization structure, infrastructure, or technology) and selecting between options based on specific goals

C1.4 Evaluating planned progress and adapting means based upon feedback

C1.5 Searching and selecting contacts, clients, and partners based upon predefined plans

**C2 Attitude towards unexpected events: avoid**

C2.1 Feeling threatened by unexpected events, therefore working in isolation with external environments as much as possible)

C2.2 Carrying out plans as defined in cases of unforeseen developments (avoid changes)

C2.3 In cases of unforeseen changes, focusing on activities within startups rather than engaging in environmental factors

C2.4 Pulling away from the project or resolving it quickly in cases of unforeseen developments

**C3 Attitude towards outsiders: competitive analysis**

C3.1 Acquiring resources through market transactions or contract-based agreements with stakeholders

C3.2 Creating and carrying out the patent strategy

C3.3 Carrying out competitor analysis and competitive positioning

**C4 View of risk and resources: expected returns**

C4.1 Maximizing personal profit

C4.2 Calculating and evaluating expected outcomes and returns

C4.3 Planning development in big steps and with large sums (including large recruitment, where *large* is relative for each company)

C4.4 Postponing stakeholder (including clients) contact at the expense of own funds (focus on internal development)

C4.5 Search for stakeholders that commit the amounts necessary for the execution of the plan

To generate initial codes, the first and second authors applied a descriptive coding technique to identify entrepreneurial logic dimensions systematically across all cases (Runeson and Höst, 2009). Descriptive coding helped organize and group similar data into categories, which was the first step towards creating themes. All events in each case were coded according to four effectual and four causal dimensions; thus, effectual, and causal logic could co-occur in the same event. The number of quotes per code is presented in Table 6. We counted how many effectuation dimensions (potentially ranging from 0 to 4) and how many causation dimensions (potentially ranging from 0 to 4) were coded per event. If at least one effectuation or causation dimension was coded for each event, we could identify that event's entrepreneurial logic.

Table 6: Number of quotes per code

Themes		No. of quotes
C1.1	Base actions upon expectations and predictions	25
C1.2	Defining and pursuing project goals	34

C2.1	Carefully interacting with environment for secrecy reasons (feel threatened by unexpected events, therefore work in isolation as much as possible)	12
C2.2	Carrying out plans as defined in cases of unforeseen developments	15
C2.3	In cases of unforeseen changes, focusing on activities within startups rather than engaging in environmental factors.	11
C2.4	Drawing back from project or quickly resolving in cases of unforeseen developments	16
C3	Attitude towards outsiders. competitive analysis	19
C4.2	Calculating and evaluating expected outcomes	9
C4	View of risk and resources: expected returns	5
C4.3	Planning development in big steps and with large sums	26
C4.5	the execution of the plan	9
E1	Basis for acting Means-oriented	49
E1.1	Building on own knowledge base and other available existing own resources	15
E1.2	Short-term planning	61
E1.3	Local infrastructure and inside environment	12
E1.4	Following personal preferences	71
E2.1	Accepting, gathering and incorporating unexpected, leading to pivots	56
E2.2	Changing and adapting any potential plans made to accommodate unforeseen events	26
E2.3	Actively exposing to outside influences, while being open minded	33
E3.1	Reaching trust-based flexible stakeholder agreements and commitments	20
E3.2	Co-create business with stakeholders	24
E3.4	Exposing MVPs to potential clients early on	27
E4	View of risk and resources: affordable loss	17
E4.1	Be willing to make affordable personal sacrifice (including non- monetary)	9
E4.2	Finding unused resources in local environment (including subsidies)	9
E4.3	Investing limited, small amounts of personal money, time and effort	21

We then counted the number of events belonging to either effectuation or causation for each case. In total, we coded 631 events from all cases. The number of coded events varied significantly among cases, from 4 to 26 events in a single case. This variance is due to the relevancy of cases and events in each case to entrepreneurial logic. The total number of effectuation codes is 450 (71.4%). The total number of causation codes is 181 (28.6%).

	A : P. Product Developm...	B : P1. Requirement En...	C : P2-Prototyping	D : P4. Implementation (...	E : P6. Man	Node Matrix
1 : C1.2. Defining and pursuing project goals	0	0	0	0	0	0
2 : C2.1. Carefully interacting with environm...	0	0	0	0	0	0
3 : C3. Attitude towards outsiders. competitiv...	0	0	0	0	0	0
4 : C4.2. Calculating and evaluating expecte...	0	0	0	0	0	0
5 : C4.3. Planning development in big steps ...	0	0	0	0	0	0
6 : C4.5. Search for stakeholders that commi...	0	0	0	0	0	0
7 : E1. Basis for taking action Mean oriented	0	0	0	0	0	0
8 : E1.1. Building on own knowledge base a...	1	1	1	0	0	0
9 : E1.2. Short-term planning	0	0	0	0	0	0
10 : E1.3. Local infrastructure and inside env...	1	0	0	0	0	0
11 : E1.4. Following personal preferences	0	0	0	0	0	0
12 : E1.5. Building on existing network	0	0	0	0	0	0
13 : E2.1. Accepting, gathering and incorpor...	0	0	0	0	0	0
14 : E2.2. Changing and adapting any potent...	0	0	0	0	0	0
15 : E2.2. Changing and adapting any potent...	0	0	0	0	0	0
16 : E2.3. Actively exposing to outside influe...	0	0	0	0	0	0
17 : E3.1. Reaching trust-based flexible stak...	0	0	0	0	0	0
18 : E3.2. Co-create business with stakehold...	0	0	0	0	0	0
19 : E3.4. Exposing MVPs to potential clients...	0	0	0	0	0	0

Figure 5 Mapping entrepreneurial logics and SE activities

### 3.4.3. Mapping Entrepreneurial Logics and SE Activities

Not every event relates to SE. We went through each case and identified the quotes that had both SE labels and entrepreneurial logic labels. An example of how two layers of codes are matched is shown in Table 7, with some quotes extracted from case S12. To understand how quotes about SE activities are related to entrepreneurial logic, we employed axial coding (Corbin and Strauss, 1990) to map them into entrepreneurial logic coding scheme.

Table 7 Startup behavior quotes from case S12

SE Area	Quotes	Entrepreneurial Logic
---------	--------	-----------------------



P1. Requirement Engineering	Either we solve them by providing them different products or we do ignore parts of the market. We make a very active statement on what kind of requirements we do fulfill. Then we turn down clients that do not believe the [00:17:48] requirement. We make a very clear statement to what we think the future of journalism is, then we pursue it. The cost of that is neglecting parts of our market.	C4.5. Search for stakeholders that commit the amounts necessary for the execution of the plan
P1. Requirement Engineering	That is because we are in a very challenging market with changing requirements, so that is what they want. Then, as we got bigger, we tried to create a more complex organization within the company. That was the biggest challenge, or at least to us, because we did not know how to do it.	E1. Basis for acting: means-oriented
P1. Requirement Engineering	There will always be requirements arriving, that is one thing. Sometimes the new requirements disrupt the old requirements. At the moment, we are working to disrupt the old products. To reinvent them and to kick the [00:15:36] away under our old products.	E2.2. Changing and adapting any potential plans made to accommodate unforeseen events
P6. Process Management	Yes, we have always been working in an agile Way. We are not adhering to any specific agile approaches, but we can't do long-term specifications. That is not doable in an industry that is changing very rapidly. We have always been working with long-term visions but with short-term specifications. The way we developed specifications, it is always with the collaboration with the clients or the customers	E3.4. Exposing MVPs to potential clients early on
P3. Software Construction	We do all software development in-house, we do not do any outsourcing to India or other places for the simple reasons that everything we do is very short cycle. It's very innovation-oriented, so our software developers probably taught 50% of the time and code 50% of the time, so outsourcing wouldn't really work for the way we work	E1.1. Building on their knowledge base and other available existing owned resources

Two-dimensional queries were created in NVivo version 12 to map the entrepreneurial logics and SE activities, as shown in Figure 5.<sup>4</sup> To aid the mapping process, we developed a qualitative codebook that includes all quotes and their associated codes (illustrated by Figure 6). Two authors read the codes, the case context and assign an explanation to them. We use collaborative notes and mind maps as additional tools to record any discoveries in the data.

#### 3.4.4. Effectuation Index for Cases

To determine whether a case is either effectuation or causation dominant, we defined an Effectuation Index (EI), which has been used in a previous study (McKelvie et al., 2020):

$$EI = X(\text{EffectuationEvents}) / X(\text{Events}) \quad (1)$$

For comparison among cases, we defined three categories based on the value of EI:

- EI between 0.7 and 1: effectuation dominant

<sup>4</sup> <https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home>

- EI between 0.3 and 0: causation dominant
- EI between 0.31 and 0.69: mixed

Category	Logic	Item	Description	Quote
Requirement En Causation		Defining new features	New features at the next stage is to add health data to the system... then we charge you because we store it for you and make sure that it can arrive. You have a premium subscription...	Normir took a short time because we used some of what we have development in the horizontal market and used it for B2B market (S15)
		Dropping new requirements	Requirements or even the basics of the technology, i.e. the promises, or potential promises of the technology, and how it was kind of perceived that it could come together, we could...	The next project is a self service gift card shop for some of our customers they basically use the ticketing system as a gift card shop. In the end I think now we are...
		Plan-driven analysis	Startups are not	... we sketched the designs of all ideas we had. We discussed one by one. The most convincing idea were selected for further development. ... I know the food delivery like that pretty well so I am often the winner... (S04)
		Requirement negotiation	Customers can	We were initially very nervous about the community potentially making this decision back on us. But the community turned out to be quite supportive. While not... That's sort of positive problem because you can't address all the possible needs at once so you need to pick the most promising ones first (S36)
		Effectuation	Engaging in stakeholder dialogues	We are in close dialogue with the gas suppliers and work closely with them, and also receive a good confirmation from them. We are running a pilot project with a... We can utilise that, yeah. But, at first when we started our development I had that experience from my trainee period. Then I listened many complaints about the... Matching requirements Startups adopt a... OSS is used in many architectures and for many purposes... Searching a suitable library was sometimes not so easy but the time was paid back at the end... Prototypes are created often for communicating with external stakeholders

Figure 6 An example of the qualitative codebook

Table 8: Effectuation Index value of each case

Id	No. Events	No. Effectuation.	No. Causes.	Effectuation Index	Logics type
S01	8	7	1	0,87	Effectuation
S02	13	9	4	0,69	Mixed
S03	21	15	6	0,71	Effectuation
S04	19	14	5	0,74	Effectuation
S05	7	6	1	0,86	Effectuation
S06	12	5	7	0,42	Mixed
S07	7	6	1	0,86	Effectuation
S08	8	7	1	0,87	Effectuation
S09	14	6	8	0,43	Mixed
S10	8	7	1	0,87	Effectuation
S11	19	14	5	0,74	Effectuation
S12	6	5	1	0,83	Effectuation
S13	15	9	6	0,6	Mixed
S14	16	11	5	0,69	Mixed
S15	16	14	2	0,87	Effectuation
S16	11	8	3	0,73	Effectuation
S17	11	7	4	0,64	Mixed
S18	12	9	3	0,75	Effectuation
S19	23	16	7	0,7	Effectuation
S20	26	13	13	0,5	Mixed
S21	11	10	1	0,91	Effectuation
S22	11	9	2	0,82	Effectuation
S23	13	11	2	0,85	Effectuation
S24	14	6	8	0,43	Mixed
S25	31	21	10	0,68	Mixed
S26	13	13	1	0,92	Effectuation
S27	6	5	1	0,83	Effectuation
S28	33	27	6	0,82	Effectuation
S29	31	25	6	0,81	Effectuation
S30	23	18	5	0,78	Effectuation
S31	25	19	6	0,76	Effectuation
S32	18	13	5	0,72	Effectuation
S33	13	7	6	0,54	Mixed
S34	6	5	1	0,83	Effectuation
S35	16	13	3	0,81	Effectuation
S36	15	5	10	0,33	Mixed
S37	18	8	10	0,44	Mixed

S38	14	13	1	0,93	Effectuation Effectuation Mixed
S39	21	18	3	0,86	
S40	27	16	11	0,59	

The EI value for each case is given in Table 8. The higher effectuation index value is, the more dominant effectual activities are found. Table 9 describes the mean EI value for startups regarding their locations (international or Nordic startups), stages (pre, startup, or post phase), and application domain sectors. Pearson chi-squared test shows no significant difference in the distribution of EI values across these categories.

Table 9: EI values among startups in different locations, stages and industry domains

Context factors	N	No Effectuation per case	No Causation Per case	Mean EI	Chi square test
<b>Locations</b>					
International	8	10	3.38	0.76	p-value= 0.80
Nordic	32	11.56	4.84	0.72	
<b>Stages</b>					
Pre-startup	14	12	4.64	0.74	p-value= 0.43
Startup	22	11.41	4.04	0.75	
Post-startup	4	7.75	7	0.51	
<b>Industry domains</b>					
Agriculture, Forestry, Fishing and Hunting	3	9	1.33	0.87	p-value= 0.12
Arts, Entertainment and Recreation	1	7	1	0.87	
Construction	1	14	5	0.74	
Education	4	7.75	6.5	0.60	
Health Care and Social Assistance	7	14.86	5.29	0.71	
Manufacturing	2	16	3.5	0.84	
Professional, Scientific, and Technical Services	18	11.33	4.33	0.73	
Transportation and Warehousing	2	11.5	8	0.57	
Utilities	2	10.5	4	0.72	

#### 4. Results and Analysis

The sections below present the data obtained during the study and how this data answers the research questions.

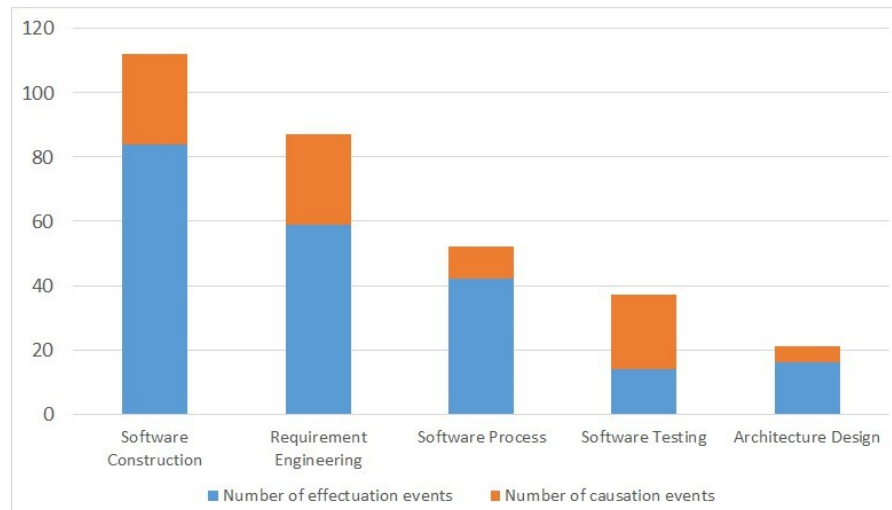


Figure 7 The occurrences of entrepreneurial logics across SE area of activities

#### 4.1. RQ1: How do entrepreneurial logics apply to SE activities in startups?

We identified many SE events tagged with entrepreneurial logic. Figure 7 presents the distribution of these events across SE knowledge areas and logic types. We observed the largest number of entrepreneurial events associated with software constructions, followed by requirement engineering, software process, software testing, architecture design, and software maintenance (details are given in Table 10). As our interviewees did not describe their SE activities to the same extent (some focused more on requirement engineering aspects, some talked more about their software processes), the numbers do not represent close relationships among the SE knowledge areas. However, we can see that both effectuation and causation logic occur in all SE knowledge areas. Effectuation logic is the dominant logic in software construction, requirement engineering, architecture design, software maintenance, and software. Testing is the only type of activity where our cases reported that the number of causation events was larger than the number of effectuation-driven events. We present detailed observations in the following sub-sections.

Table 10: Entrepreneurial logics across SE activities

Area	No. SE Events	No. Effectuation Events	No. Causation Events
Software Construction	112	84	28
Requirement Engineering	87	59	28
Software Process	52	42	10
Software Testing	37	14	23
Architecture Design	21	16	5
Software Maintenance	15	9	6

##### 4.1.1. Requirements Engineering

The thematic codes for entrepreneurial logics in Requirement Engineering are given in Table 11. Requirement engineering activities include the elicitation, analysis, validation, documentation, and scoping of software requirements (Bourque and Fairley, 2014). In many software startups, requirements based on hypothesized business or market demands and the process of requirement elicitation were directly associated with the customer development journey (Blank, 2013; Melegati et al., 2019). Requirement elicitation and management in many software startups can be characterized by effectual logic. The primary sources of requirements are internal stakeholders (i.e., the startup founders) and external stakeholders who can be reached with the existing resources, i.e., the entrepreneurs' personal and professional networks. It is common for startup founders to generate product ideas themselves or based on internal knowledge and then derive concrete requirements by engaging customers early in the development process. This differs from the situation in established companies, where several requirements originate from paying customers or marketing departments.

We actively looked for the right requirements from our customers. We occasionally searched for any product area. Recently this year, we started a new product feature, and this disrupted one of our old features.

That decision was made in collaboration with our customers and was implemented with their sponsorship. It is a collaboration agreement in which they finance parts of the development (S14).

In market-driven startups, requirement engineering is often centralized around lead users (von Hippel, 1986), who can express demands currently unknown to the public. They are not the source of requirements, features, and main factors driving startup learning. They contribute to product brainstorming, testing, and feedback or even participate in developing and co-creating new products or services. We found that startups often collaborate closely with some lead users or include them as the startup teams' internal members.

We sketched the designs of all the ideas we had. We discussed them one by one. The most convincing idea was selected for further development. . . I know the food delivery and stuff like that pretty well, so I am often the winner (S04).

We see that startups might not find many more requirements for their products than what can be collected from their lead users. Startups might start with a sub-optimal set of requirements due to the limited inclusion of lead users (Nguyen-Duc et al., 2017), and the requirement list gradually evolves due to changes in startups' resources and networking positions. It is also possible that not all users' inputs manifest as valuable for customers or a market. Startups often have a low threshold for stakeholder participation and influence on their businesses and products. Depending on internal social capital, startups might involve lead users to different extents. The low threshold seems to be a challenge for startups to identify valuable inputs, and hence, there could be a long journey to identify the winning features.

Moreover, startups often build prototypes as a means of communicating their requirement engineering activities. However, prototype features, such as levels of fidelity and types of prototypes, depend highly on the currently available resources for making such prototypes. For instance, the CEO of S04 mentioned using screenshots that she created in communications with her first customers:

We worked directly with a customer's organization and learned their current solutions. We described our approach using prototypes like screenshots. It would be hard for them to realize the benefit without concrete examples... (S04).

The prioritization of requirements that match the current development resources is more evidence of effectual logic. Startups might have many innovative ideas, but they will often produce low-fidelity prototypes due to the lack of development resources. Functioning prototypes (MVPs) are often built based on available software components, libraries, frameworks, and even other software products. For instance, many pre-existing libraries are used to develop AI-based software solutions (S21, S32), create new operating systems (S10, S14), or assemble web-user interfaces (S01–S04, S06–S09, S30).

Table 11 Entrepreneurial logics in requirement engineering activities

Logics	Codes	Explanations
Effectuation	Engaging in stakeholders within founders' networks	Startups utilize their existing resources and connections to identify, gather, and validate their product requirements
	Quantity and quality of requirements depend on lead users involved	Startups actively seek lead users (who deal intensively with the requirements where there is no suitable solution existing on the market) within their resources and capacity
	Significance of internal source of requirements	Requirements might significantly come from internal stakeholders
	MVPs for requirement communication	low-fidelity MVPs due to the available resource and time in the startup team
	Resource-driven requirement prioritization	Requirements might be adjusted and prioritized due to available technical resources, infrastructures, and code
	Tolerance of sudden changes in requirements	Startups tend to accept changing requirements from key customers that might lead to business-driven pivots
	Adaptive approaches to prioritize the requirement	Requirement selection and prioritization depend on the business context, iteratively leading to a family of product line

Causation	Requirements extracted from a comparative analysis	Product features are identified via successful experience or competitors' products
	Plan-driven analysis of markets, customers, and competitors	Startups are not able to develop all collected requirements, and a formal prioritization process is often needed
	Avoiding changes to core business value development	Startups often negotiate with customers on requirements that are not aligned with their core business values. Negotiation is often done based on a long-term goal analysis
	Dropping new requirements to avoid unforeseen development	Requirements or new features that lead to an unforeseen cost-benefit situation are canceled

Regarding requirement change management, software companies in the early stages (pre-startup or startup) tend to accept and incorporate significant changes in their feature list, leading to a pivot. This can be explained by the effectual logic attitude towards unexpected events.

It's very difficult to say no when "giant" customers tell you we need that functionality. If you're going to have us as customers, you will have to make it. We need it in the contract that you have to make it. We also built it [the software product], and we built it bigger and bigger (S11).

The change can also lead to reworking during product development, which startups will need to cope with:

There will always be new requirements arriving, that is one thing. Sometimes the new requirements disrupt the old ones. At the moment, we are working to disrupt the whole old product, reinvent them, and throw away the whole codebase (S14).

Startups often adopt adaptive approaches to deal with changes in requirements. Requirement selection and prioritization depend on the business context and, iteratively, lead to a product line family. To sum up, many requirement engineering activities are recorded in the association to effectual logic.

Software startups also express causal logic when dealing with requirement engineering. Goal-oriented requirement engineering occurs when defining and planning user stories with limited uncertainties. The analysis of customers, markets, and competitors is goal-oriented and follows some kind of predefined plan. Requirements are then turned into short-term sprint backlogs and are often implemented according to the sprint plans.

We found out that in Norway, the public only knows one ticketing provider called Company A, which is owned by Company B, which is owned by Company C, a big international company. Then we saw a market possibility for providing a ticketing system, a DIY ticketing system for the smaller venues because Company B and Company C, everything you had to do you did by email correspondence (S11).

When faced with an unexpected change—for example, customers proposing an innovative but peripheral requirement—many startups implement a causation-centric strategy by avoiding unforeseen consequences in developing a product or business, even though this would lead to some customers' requests being adjusted or even dropped.

We turned down clients that did not believe the [00:17:48] requirement. We make a very clear statement to what we think the future of journalism is, then we pursue that, and the cost of that is neglecting parts of our market (S14).

Observation 1: Requirement elicitation, negotiation, and management tend to be effectuation-driven processes in which startups explore new types of products or customers. Certain finely detailed activities, such as requirement breakdown, estimation, analysis, and validation, tend to be causation-driven when requirements are known to some extent.

#### 4.1.2. Software Construction

The thematic codes for entrepreneurial logics in Software Construction are given in Table 12. Software construction is the creation of working software through a combination of configuration, coding, unit testing, and

debugging (Bourque and Fairley, 2014). In software startups, construction activities apply to both final software products and MVPs. Concerning effectual logic, MVPs are typically developed according to this model in terms of how the speed of implementation, the functionalities, and quality of MVPs mainly rely on the available technical competence in the startup companies. A startup can launch a good front-end prototype very quickly with a user experience expert in the team. A startup can also start with low-fidelity wireframes created by a startup founder who does not have technical competence. The founders of the startups examined in this study created various MVPs, including paper sketches, mockup design tools, and competitor products (S2, S09, S11, S13).

The first version is like a hack; it took a lot of time to make it up and running. It was impossible for teachers to use because it needed a developer to set up all the network things. It was done in a really hacking manner. Also, it was one instance so it could run one quiz at a time... It failed completely as we just had to throw away the prototype (S09).

The underlying logic is to accept development waste and focus on learning from throw-away prototypes. The minimum effort could also become a wasted effort, for example, when the prototype simulates but does not illustrate. The CEO of S11 introduced the concept of faking a product: “fake it until you make it.” Without technical capacities, he demonstrated his business vision with a “faked product,” which implies a lack of primary quality, both in terms of functionality and user experience. The CEO of S02 expressed that what they built, in the beginning, is a minimum potential prototype, but not the MVP:

Building a prototype is like building a fake house. The exterior design is done, you can see how it feels, but the internal part is empty. It helps you figure out if this kind of house you want to live in... We are creating a minimum viable product, not a completely viable one (S02).

Consequently, road mapping and planning for MVPs are often overlooked. After completing an MVP, the creation of the next one is often decided opportunistically. Essential elements of a plan are often neglected: for example, how many MVPs are needed, for what the next MVP will be used, and the criteria for evaluating MVP learning outcomes. Startups accept failure when building MVPs and embrace exploratory development at the cost of economic sacrifices.

Startups are also known to adopt workaround solutions, such as a set of files that is reasonably functional. Developing and testing MVPs at fast rates enables startups to validate their assumptions about their business viability. However, the focus on development speed can also lead to minimum viability. A workaround solution is different from a planned temporary solution, i.e., a piecemeal MVP (Ries, 2011). In some cases, startups have to throw away an MVP that was not designed for long-term use. Startups often aim to develop software early by incrementally adding features into the prototype (Nguyen-Duc et al., 2017). In this way, TD is created as startups focus on speed and neglect quality (Giardino et al., 2016). More seriously, some software products were built using architecture not designed for scale. S09 developed an in-class quiz-based application to check student understanding of lectures in real time; multiple prototypes developed by the CEO or as a student project experimented with different classes. The final prototype, which captured refined design and business ideas, was further developed into a version suitable for launch; however, the release’s quality did not match the performance demanded by unexpected growth in users.

In another case, S27 rapidly developed an initial MVP with a hastily built front-end and a hacking back-end function with no security that just achieved minimum performance. The MVP was thrown away, and the company acquired its first seed investment for serious prototype development. Somehow, the launched product contains many components from previous MVPs with a large amount of technical debts. The team deployed the product to customers and extended it further. This later was perceived as a mistake that costed the company significantly:

At one stage, you just had to drop everything but keep the concept and create it from scratch. The concept was good, the implementation was not bad, but it didn’t fit into the commercial world. And at another stage, we needed to get new people with some new minds that could think slightly differently (S27).

Many startups hired external resources, such as local contractors or offshore software vendors, based on their experience and networks. This practice is often the case with non-technical founders or companies with limited in-house technical competence. Examples of local contractors are consultant companies, makers’ spaces, student projects, and freelancers. In case S15, skilled contractors were hired to achieve a quick start with a functional MVP. As mentioned by the CEO, the use of external resources enables speedy product experimentation and

development. Furthermore, as contractors are not an integral part of the startup, their relatively easy dismissal facilitates scaling-down activities that may be necessary if the startup lacks funding or changes directions. Some startups use local contractors, while others hire offshore vendors. Making use of local vendors can be a feasible option:

The local one [vendor] delivered very quickly. It is critical that the component from China comes on time, especially when we needed to demonstrate in the week after in the UK. It is always a matter of time. If we could do everything internally, we would have saved a lot of time sending; it would have been great! (S38)

Table 12 Entrepreneurial logics in software construction activities

Logics	Codes	Explanations
Effectuation	Resource-based MVP development	MVP development relies on existing and accessible technical competence
	Product experiments with tolerance for waste	MVPs are created for demonstration, which is not suitable for long-term use and often thrown away in the later stages of the startup journey
	Overlooked product road-mapping and planning	Construction of MVPs usually occurs in an experimental and opportunistic manner
	Speed-first MVP development	Startups often need to balance quality and speed to market, and in most cases, time-to-market is prioritized
	Recruitment of external competence	Hired developers or contractors are often from the founders' network
	Component-based development	MVPs typically contain a significant number of ready-to-use components that can be plug-and-play in a short time
	Innovative product development requires exploratory approaches	Innovative products often involve RandD activities that are not purely driven by goals and plans
Causation	Short-term plan-based product development	Product development is planned from the requirement to launch, and different efforts are performed to achieve the initial plan
	Preventing software constructions from business threats	Software construction might be paused because of financial and business uncertainty

Most of our startups leveraged existing libraries, frameworks, and components to build a runnable MVP quickly, accessing either paid APIs or Open Source Software (OSS) components. Particularly, the adoption of OSS components was mentioned in all the cases, from using OSS tools (S19) to the integration of OSS code (S02, S03, S05, S20) to participation in the OSS community (S18). The main benefits, including reduced development cost and faster time-to-release, were mentioned by the CTO of (S19) and (S20):

The things we are doing today, we might not even come to the idea of making it happen if we do not have open-source software (OSS) as an experiment. Without OSS, it would take a lot of time and be very costly (S19).

It is very hard nowadays not to use OSS artifacts, especially when with Android development (S20).

It was observed that component-based development can influence the product architecture of early MVPs. In S02, OSS JavaScript frameworks were considered the central part of product architecture for the web and mobile applications. It also appears that many advanced technologies were adopted using OSS:

A core part of our product includes a machine learning (ML) algorithm. We are lucky enough to find ML libraries in C++, and they are entirely OSS (S02).

One possible challenge of using ready-made components is to find a suitable component in terms of maturity, code quality, and level of support, which also appears as an effectuation-driven behavior:

OSS is used in many architectures and for many purposes... Searching a suitable library was sometimes not so easy but the time was paid back at the end (S05).



The selection process needs to consider functional requirements and quality requirements for the component and the whole product. The CEO of S12 stated that many large companies had offered free APIs to access their data and functionality, integrating them into final software products to consider other issues, such as quality, scalability, and cost. OSS components might not be the minimum available solution, but they reduce the inherent risks of scaling for later phases.

In terms of causal logic, from a short-term perspective, software constructions are plan-based, with concrete expected outputs. Startups also adopt a plan-based product development approach from the requirement elicitation activities to product deployment. So long as the product requirements are identified (e.g., an established sprint backlog), it is expected that the sprint will be operated without many changes.

For us, this was not the major change because the product was ready, and the customer had the need for it. It was a pretty straightforward delivery for us (S14).

Business and financial stability appear to be important influencing factors on whether causal logic occurs and underlies the software construction process, as in case S03 with secured initial seed funding:

For the first two or three years, we have been only a product development-driven company. Everything we did was product development... We have grown a lot in Norway by product development, word of mouth, and customer satisfaction (S03).

Startups also express their causation-driven behaviors by avoiding unexpected events and focusing on internal project activities rather than engaging in external interaction. For instance, S25 had to stop its development activities due to uncertain financial conditions in its early stage.

The operations... stopped, like, one-and-a-half years ago, when we noticed that we were not capable of raising the risk funding for the development of the required technology (S25).

Observation 2: MVP development is typically an experimental and waste-tolerant process, driven by time-to-market, available competence, and internal incentives. Software construction is often opportunistic, and plan-driven coding activities occur in the short term or later stages of a startup's life cycle.

Table 13 Entrepreneurial logics in software design activities

Logics	Codes	Explanations
Effectuation	Early customer involvement in solution design	Startup actively involves customers in the design space to co-create business value
	Solution design as an experimental process	Solutions for a given customer or market are iteratively visualized through experimental activities
	An adaptive approach for a configurable product design	The product design in some cases needs to be adaptable to different customers' requirements
Causation	Technical architecture as an optimizable task	Architectural decisions are made with a thorough consideration of cost-benefit trade-offs

#### 4.1.3. Software Design

The thematic codes for entrepreneurial logics in Software Design are given in Table 13. Software design represents the problem-solving space where actual business value is planned to be implemented. Software design can include both user interface design and architectural design (Bourque and Fairley, 2014). The effectual logic is apparent in the solution design process, i.e., identifying the best solution for a current customer or market demands. This process might be experimental, means-driven, and change-prone. The process often involves early customers. In S19, the startup not only exposed their MVPs to potential customers quite early, but also used MVPs to involve the customers to their design process:

Yes, I think it is important to get the customer involvement in the [product] design... Otherwise, it would be a bit scary to launch a new system with assumptions that someone would use it (S19).

Regarding architectural issues as part of the experimental process, S37 said that:

We have struggled with the choice of platform for the autopilot. Controllers need to be implemented on something, so we have spent a lot of time on embedded components to get the right protocols to control the reserves and get to know what ran on the OSS stuff (S37).

Because of active customer involvement, in some cases, architecture needs to be adapted to cover different requirements:

Because there are many other parties involved. And many other systems where the interfaces might not be so able to integrate if they are old legacy systems. So they are usually the biggest challenge (S31).

We also observed a customized software design when different requirements arose in the design phase. The architecture for a single product might need to evolve into a product-line architecture with extendibility.

In terms of causal logic, technically speaking, software design is a plan-driven activity. The integration of complexity and other quality attributes in functional software is an achievable task. Architectural decisions are made with a thorough consideration of cost-benefit trade-offs, especially when a system needs some quality attributes for the long run, i.e., performance, and availability.

Observation 3: From a business perspective, software designing is an effectuation-driven process; from a technical perspective, software designing could be plan-based and optimizable.

#### **4.1.4. Software Testing**

The thematic codes for entrepreneurial logics in Software Testing are given in Table 14. Software testing includes four levels: unit, integration, system, and acceptance testing (Bourque and Fairley, 2014). Regardless of levels of testing, we focused on the testing activities that evolve potential or actual users. Product testing is based on assumptions and hypotheses set by the startup about generated value for users and customers. In this sense, product testing is an important mechanism to validate the product/market fit. Many startups do not talk about their testing process in detail. From our observations, startups appear to have more causation-driven testing activities than effectuation-driven ones, as described below.

Effectual logic appears in software testing as the minimum viable testing concept. As found in previous studies (Giardino et al., 2016), software startups prioritize time-to-market over acceptable product quality. This practice is represented by the lack of proper test plans and insufficient testing at different levels. Startups use existing developer resources, such as spare development time, in their milestone-driven plans for testing.

We prefer to work quickly, and writing tests could double the development time... If these parts are built to be replaced later, then we think there's no point in spending time on testing (S2).

They can tolerate possible losses due to the lack of quality focus. The available resources and equipment then influence the testing activities:

We are working with a partner to put in place some equipment for further testing, but until now, we have focused primarily on approximate measurements due to a lack of premises and equipment (S39).

When releasing and testing a product version to early adopters, a company may sell the product to others through word of mouth. This fits with the effectual logic concept of "initial customers as partners and vice versa." Overall, it seems that startups with a broad base of potential customers and investors interested in what they are doing as they develop a new product or service have an advantage over those entrepreneurs operating in isolation.

Causal logic is more apparent in the development of particular types of products (e.g., hardware-relevant products) or specific application domains (e.g., automotive, and healthcare industry), where quality is intrinsic to a released

software system. In the mindset of these startup founders, testing is as essential as implementation. High quality in hardware development is vital because of the cost associated with production and quality mistakes, which dramatically affect the perceived functionality of the product (Berg et al. 2020). In contrast to software products, it is challenging to implement changes and improve the quality after the product has been produced and assembled.

Failures can cause high costs, more work, and, at worst, a security issue to make sure no one gets hit if it [a flying drone] falls. This is opposed to a car or a boat because testing them is much easier. Setting up robust tests and making a foundation for testing for something that can fail in the air is a unique challenge with flying things. The quality has to be better, and it is not easy to test things... We must accept that the fastest way isn't always the best one. For flying, it is important to do things properly instead of choosing quick solutions (S37).

Some companies decided to do test-driven development by developing both requirement description and test cases, using the test cases to track the software development, as illustrated in case S40:

Test-driven development is also changing now a little towards acceptance of test-driven development. So, we can write tests that customers can also read and verify by themselves that they are passing and that we are implementing the right features. Also, we are moving more towards automated end-to-end tests, that the test begins from the user interface and ends... (S40).

Observation 4: While system testing and user acceptance testing are often results of causation-driven behaviors, effectuation-driven testing is often applicable for demonstration.

Table 14 Entrepreneurial logics in software testing activities

Logics	Codes	Explanations
Effectuation	Minimal viable testing	Startups might perform testing just enough for purposes of demonstration or launching
Causation	Testing is by-designed in specific types of products	Hardware-related products often require heavy upfront testing
	User acceptance of test-driven development	Startups with quality as value proposition need to achieve their plans for user acceptance tests
	Test-driven development	Test plan are often made at the same time with requirement specifications in hardware-related products

#### 4.1.5. Software Maintenance

The thematic codes for entrepreneurial logics in Software Maintenance are given in Table 15. Software maintenance in SE is about modifying software products after delivery to correct faults and improve performance or other attributes (Bourque and Fairley, 2014). In startups, software maintenance and construction are often mixed when providing running software for some customers and, at the same time, developing new features or new variants of the product. Where effectual logic is concerned, startups often take on many maintenance tasks as they support their first bespoke customers. In these cases, product improvement and new features are not typically planned, and customer satisfaction is an important criterion that directs further development.

Question: Have you planned for a way to upgrade the software of sold gloves?

Answer: We have not thought about that. We have assumed that if the user thinks there is something wrong, then the user will contact us. Then we help the user with the error that has occurred. We have no analytical overview of the products that are out there (S40).

Table 15 Entrepreneurial logics in software maintenance activities

Logics	Codes	Explanations
Effectuation	Customer-driven software maintenance	Maintaining tasks tailored to specific customers
	Contingency approach of managing TDs	Depending on contexts, the debts can be managed, accepted, avoided, or ignored
	Reacting to bespoke change requests	Maintenance tasks occur from new feature requests or bug fixes for bespoken customers

Causation	Scheduled management of tools and infrastructures	Maintenance tasks, including infrastructure and configuration management, are typically scheduled and repeated
	Planned software maintenance	Software maintenance tasks are planned along with development and testing

Another trigger for effectuation-driven software maintenance is the TD incurred during software construction (Giardino et al., 2016). The metaphor implies that “interest” has to be paid during maintenance and development activities and that the “principle” should be repaid, i.e., with code refactoring, at some point for the long-term health of the software product (Krutchen et al., 2012; Seaman and Guo, 2011). While the startups agreed that TD trade-offs are crucial for their businesses, they each handled the debt differently. It is often uncertain whether the impact of “work-around solutions” on later maintenance tasks when products are operating in a customer environment.

In the beginning, we made a lot of mistakes, but they didn’t last long... Now that you started re-coding the system, leaving roughly six months of work behind... you said let’s leave it there (S11).

The effectual logic here is shown by the contingency approach that TD can be purposefully avoided, fixed, or ignored.

In terms of causal logic, we also observed that, within these startups, maintenance tasks, including infrastructure and configuration management, are typically scheduled and repeated regularly. Software maintenance can also be planned to some extent to avoid unexpected incidents and optimal in terms of cost-benefits:

It was implemented in such a way that it was not difficult to work on it or further develop it (S40).

Observation 5: Software maintenance in software startups occurs opportunistically. Dealing with TD is effectuation-driven by nature. Startups often throw away systems that are not working and develop new systems rather than reverse engineering the faulty product.

**4.1.6. Software Process**

The thematic codes for entrepreneurial logics in Software Process are given in Table 16. Startups are known to adopt a lightweight and agile workflow rather than following a specific formal method (Pantiuchina et al., 2017; Nguyen-Duc et al., 2017). Many startups do not have actual processes or a systematic way of working because they do not often prepare for a long run. Align with the effectuation approach, software startups tend to be agile or even ad-hoc and reactive:

Yes, we have always been working in an agile way. We are not adhering to any specific method, but we cannot do long-term specifications. That is not doable in an industry that is changing very rapidly. We have always been working with long-term visions but with short-term specifications. The way we developed specifications, it is always with the collaboration with the clients or the customers (S14).

Agile development was mentioned as the best approach to achieve speed and agility in startups. The CEOs related agility to less upfront planning and the short-term driven evolution of the startups. They also mentioned the speed of prototyping, development, and fast time-to-market when asked about an agile approach. Employees at the startups stated that full control of development activities and partnerships would prepare them to respond to unexpected changes. Some startups also highlighted the importance of team collaboration over defined processes. The adoption of certain agile practices or approaches might differ between the development of hardware and software elements:

Our MVP is relatively simple, while software changes likely happen all the time. We are still trying to find what is the right way to do it (S37).

Many startups characterized their workflow as a trial-and-error approach, adopted to deal with uncertainty in business and technology. It is worth noting that technological uncertainty might be due to the complexity of technology and the team’s available technical competence.

Typical sprints are anywhere between one and five days, and we always give very small steps to make sure that we don't head down a blind road, a blind alley. To make sure that we all understand what we're doing without making (S12).

Startups might not know which development approaches are practical for them due to their relatively short operation history. This is different from established companies, where they have learned and adopted stable working approaches. The journey of learning about processes and practices is rather means-driven than goal-driven; the processes are proposed by and adopted for the available resource in the startups. The effectiveness of the adopted process is then determined by the current startup team, current resources, customers, and products.

Table 16 Entrepreneurial logics in process management activities

Logics	Codes	Explanations
Effectuation	Short-term planning	Short iterations are commonly adopted with a vision of at most six months in advances
	Change-prone and dynamic development environment	Startups might face difficulties in adopting a set of specific development approaches due to the quick change of the project context
	Self-defined workflow	It is typical for many startups to adopt no formal guided development approaches
	Evolving working processes and practices	Changes in organization or product might trigger the need to try better development approaches
Causation	Plan-driven adoption of software processes	Startups might pursue a strategic goal of adopting software processes and practices

We have not made any such routines, so we are at that stage that we learn that we should do it. We started very sharply, and we have not yet reached a point where we have realized that it could help us. I know that we could probably have served more formal routines (S39).

Startups often react to their environmental contingencies by adapting their workflow to fit the new financial, organizational, and managerial conditions. Again, the means-driven attitude applies to the startups because their adjusted working approaches will depend on their internal competence and experience with methodologies. A startup team would not be likely to try out a Lean Startup approach if they do not have anyone in a team with prior experience with this style.

We came to a crossroads in February, where we decided to let one tech team continue to work on it, and one team started to work with flex sensors. We wanted to see if we could get a faster prototype by changing the solution method (S40).

In terms of causal logic, we also observed cases where plan-driven product development was adopted from the beginning. For instance, S03 had little uncertainty about their business due to the investment and precise product requirements. The product development was prioritized and planned in a year based on the formal analysis of product requirements and a stable development team:

In the next 6 to 12 months, we are going to move into a real strict agile process. Because we have our daily stand-ups and our backlogs and stuff like that, but we try to keep it a little loose (S03).

Observation 6: Software startups are characterized by self-defined, adaptive, and opportunistic workflows. The evolution of practices and processes is expected through startup development.

#### 4.2. RQ2: How do entrepreneurial logics apply to software development at the company level?

Insights from RQ1 do not give us a comparative view among startups regarding how they adopt entrepreneurial logic during their product development. We calculated the Effectuation Index (EI) using events extracted from SE activities for each startup case. There were no startups that included only causation logic or effectuation logic. To search for a possible explanation for the application of effectual logic or causation logic, we conducted a Chi-square test (as shown in Section 3.4.4) in Mean EI values across startup locations, phases and industry domains. Qualitatively, we looked at the common codes that are identified as effectuation-driven or causation-driven activities and summarized them at the company level. The type description below applies to a state of a startup, without excluding the possibility that startups shift among these types. Observations from 40 startups showed that

at a certain point in time a startup can be characterized as either an effectuation-dominant startup or a mixed-logic startup. The list of startups cases according to their types is given in Table 17.

Table 17 Entrepreneurial logics occurred in the company level

Startups type	Definition	Common conditions	Startup Cases
Effectuation-dominant	Startups that are experiencing the major number of their product development activities under effectuation logics	Great level of uncertainties Limited resources Frequent iterative processes Technical debt Pivot-ready	S01, S03, S04, S05, S07, S08, S10, S11, S12, S15, S16, S18, S19, S21, S22, S23, S26, S27, S28, S29, S30, S31, S32, S34, S35, S38, S39
Mixed	Startups that have significant number of effectuation-driven and causation-driven activities	Reduced uncertainties Acquired team competence Agile-like processes Process improvement	S02, S06, S09, S13, S14, S17, S20, S24, S25, S33, S36, S37, S40
Causation-dominant	Startups that are experiencing the major number of their product development activities under causation logics	Managable level of uncertainties Traditional software development processes	None

#### 4.2.1. Startup Type 1: Effectuation-dominant

Effectuation-dominant startups (27 out of 40 cases) often initiate with some unique advantages: for instance, a product idea that did not previously exist, a market segment with little competition, a group of talented developers, or an existing source of customers. These startups strongly emphasize personal knowledge as the starting point, i.e., the founders are competitive in business competence or technical competence. They also rely heavily on their internal resources. This can be seen from startups deriving the product requirement internally or with existing requirements (Klotins, Unterkalmsteiner, Gorschek, et al., 2019). The requirement elicitation process might involve internal or external (or both types) stakeholders (Melegati et al., 2019). These stakeholders include both lead users and individuals who commit to resources, and in many cases, startups need to discover the most valuable requirements. Social capital, the relationships between people in various networks, is critical for startups because these companies often explore their network to identify new requirements, explore business opportunities, and recruit partners or employees.

Effectuation-dominant startups also rely only on resources that they are willing to lose. Almost all of our effectuation-driven startups threw away many MVPs, even high-fidelity ones (Duv and Abrahamsson, 2016). Startups are willing to take the risk that their products or features are not desired in the market and, in most cases, are ready to pivot to a new idea if necessary (Saravathy, 2001). In these startups, technical redirection can happen any time new information is unearthed. These startups do not focus heavily on components and system testing. Software maintenance is carried on in parallel with development with a short-term focus on current customer satisfaction. From this perspective, phenomena such as TD or speed over quality are probably unavoidable and perhaps a way for startups to prepare for the possible losses incurred with a technical pivot. Effectuation is an iterative process, and startups learn continuously from their experiences. We also found that the learning process is *ad hoc* and informal, lacking knowledge discovery, extraction, and storage. This generates challenges in the future when startups scale up their products and organizations.

Observation 7: An effectuation-dominant startup focuses heavily on internal resources and social capital. The startup embraces a focus on speed over quality, neglects quality assurance investment, accepts TD, and tolerates technical pivots to deal with uncertainty.

#### 4.2.2. Startup Type 2: Mixed-logic

Compared to effectuation-dominant startups, mixed-logic startups (13 out of 40 cases) have reduced uncertainties regarding their markets, funding, and team conditions. Either they are spin-offs from established companies and inherit well-defined problems with existing customer contacts (S06, S20), they have evolved into more stable stages (S09), or they operate in a regulated domain (S40). In these startups, we found several middle-term and

long-term plans regarding their product development. These startups emphasize their plan-based analysis, selection, and prioritization of requirements to a (partly) validated market. When they implement their goals, the startups tend to ignore external influences, opportunities, and requirements and instead focus on achieving their visions. Product development is more similar to methodologies reported by established companies. Software development methodologies become more of a concern when startups look for productivity, quality, and a sustainable working experience. In these companies, the significant difference is the investment in testing activities. We also observed the adoption of formal approaches with a large upfront sum, such as test-driven development (S40). Activities such as architectural designs and software maintenance contain a lot of planning and analysis.

Observation 8: A traditional SE process and practice is relevant to a mixed-logic startup where both business and product development is relatively and subjectively predictable. A mixed-logic startup is often the continuation of an existing business or a startup at a certain maturity level.

#### **4.2.3. Startup Type 3: Causation-dominant**

Startups that are experiencing the major number of their product development activities under causation logics are not observable from our sample. By definition, these startups would adopt long-term and analytics-driven approaches. The idea type-3 startups might have an analytical approach to customer requirements, stable value propositions which unlikely to change in a short-term perspective, an early and clear overview of their product architecture, adopt principled software development processes and practices including test-driven development, continuous integration, and DevOps (Section 4.1).

## **5. Discussion**

### **5.1. Discussing the Primary Observations**

By applying entrepreneurial logics, this study explains many findings from previous software startup research. Melegati et al. (2019) reported that requirements engineering has multiple influences and helps explore the market opportunity and devise a feasible solution. We observed that requirement elicitation and negotiation tend to be effectuation-driven. Startup founders use their current knowledge about technology, markets, cultures, and social capital to identify the product's market fit. Klotin, Unterkalmsteiner, Chatzipetrou, et al. (2019) reported a survey result showing that internal sources, such as brainstorming and the invention of requirements, are the most popular requirement sources. The authors also reported challenges establishing contact with their potential customers and involving them in the product work. In our findings, we showed a means-driven principle, and the startup's social capital would likely shape this challenge. We emphasize that the real problem of requirement engineering in startups is that many customers' requirements would probably not possess the right inputs for product-killing features. Effectuation logic helps to explain why a "lean" approach suitable for User-Centered Design in software startups (Hokkanen et al., 2015). Startups in the early stages often search for lead users from their social capitals and this process is probe-and-sense without detailed plans in advance.

Many startups are not successful in learning from their MVPs due to the effectuation-driven behaviors. They tend to overlook product roadmapping and planning, reuse MVPs for many different purposes and in different scenarios (Duc and Abrahamsson, 2016), which leads to accumulated TD (Giardino et al., 2016). With the attitude of tolerating for (prototyping) failure, MVP building process is seen as a waste-tolerant process rather than a validated learning process in startup and that this process seems to be driven by time-to-market, available competence, and internal incentives. To improve the situations, lightweight guidelines at operationalization levels would be a possible approach (Bosch et al., 2013, Nguyen-duc et al., 2020).

Software testing is a particular engineering area with a dominant number of causation-driven events. On the one hand, we have observed the so-called "minimum viable testing" approach as an inappropriate quality assurance approach. On the other hand, startups often mention testing as a plan-driven endeavor. This happens in startups developing quality-critical products (e.g., hardware startups), safety, and security-critical domains (e.g., health-care and automotive), or in a startup with established software development methodology (process improvement). Klotin, Unterkalmsteiner, Chatzipetrou, et al. (2019) argued that startups could benefit from more rigorous testing practices. We can agree with this observation only when startups have quality attributes as their core value propositions. When MVPs are released for events such as demonstrations and funding pitches, testing is effectuation-driven and minimalistic. Giardino et al. (2016) pointed out several contributing factors to accumulated TDs in software startups, including lack of architectural design, automated testing, and minimal project management. These factors fit well with effectual logic as startup members are prepared for changes,

associate change with affordable loss, and avoid excessive upfront investment in design, implementation, and testing. Throw-away work seems to be a natural part of the startup, and reuse seems to be opportunistic.

Regarding software processes, software startups can be characterized by their opportunistic workflows. We would argue that the entrepreneurial logic would determine which software processes and practices are adopted. Pantuichina et al. (2017) reported that startups adopt agile practices differently and communication practices, such as daily standup meetings, are not common among startup teams. Their findings of the overwhelming of speed-related practices reflect the adoption of effectuation logic in software construction and testing. Klotin, Unterkalmsteiner, Chatzipetrou, et al. (2019) showed that start-ups often have communication issues, shortages of the domain, and engineering expertise. From the view of effectuation logics, startups tend to include any people who can contribute to their value proposition. This can solve the need for competence and knowledge in a short time but poses a challenge of team cohesiveness. While a team often needs to go through several steps to reach their optimal performance (Tuckman, 1965), startups might have a challenge reaching this performing stage.

Our findings propose a classification of startups into either Type One (effectuation-dominant), Type Two (mixed-logic) or Type Three (causation-dominant). Once startups are organized into distinct types, it will be easier to reason about their engineering activity decisions. Traditional SE processes and practices are more relevant to causation-domain startups (Type Three) or mixed-logic startups (Type Two), where both business and product development are relatively and subjectively predictable. This aligns with some observations from previous studies, i.e., startups in the early stages adopt less formal management practices than those in the post-startup stage (Klotin, Unterkalmsteiner, Chatzipetrou, et al., 2019). We argue that a startup classification is a contingency approach, in which both internal factors (e.g., founders' experience and characteristics, team competence, and available technologies) and external factors (e.g., institutional factors, startup ecosystems, and market conditions) might lead to startups becoming one of the three types. The cross-sectional view on startup classification does not mean a consistent occurrence of the logics in all engineering activities, e.g. causal logics can still be observed from effectuation-dominant startups. Hence, we suggest the reasoning for startup tactics should be done in the association with SE decisions to be taken and its contexts. Besides, we do not capture the dynamic aspect of startup types, i.e. startups might be in a transition from Type One to Type Two or from Type Two to Type Three. A study of logic shifts, as shown in entrepreneurship research (Reymen et al., 2017), needs to be done in a longitudinal manner.

## 5.2. The Applicability of Entrepreneurial Logics in Software Startup Engineering

Exploring and comparing causation and effectuation logic to make sense of startups' business activities are widespread in business research (Sarasvathy, 2001; Sarasvathy and Dew, 2005a; Chandler et al., 2011; Reymen et al., 2015; Reymen et al., 2017; Harms and Schiele, 2012; Smolka et al., 2016). Sarasvathy (2001) emphasizes that causation logic is more suitable for existing markets, and effectuation logic is more suitable for new markets and products. Chandler et al. (2011) illustrate the occurrence of these logics in business experimentation. Flexibility regarding unforeseeable events in effectuation has been contrasted with carrying out a planned strategy under causal logic (Reymen et al., 2015). Our observations show that both kinds of logic can be found in different SE areas of activities. After comparing our results with those of previous studies (Giardino et al., 2016; Melegati et al., 2019; Klotins et al., 2019; Tripathi et al., 2019), we understand the logics behind how some engineering activities are carried on. Indeed, after quantifying the number of logic-driven events across 40 startups, we did not observe any case adhering to only one logic. We do not have enough insight into each case to conclude possible patterns of adopting logics, engineering activities, and their consequences; however, effectuation logic appears to be the primary principle behind product development in the early stages of startups. Moreover, as SE deals with a systematic development approach, we would expect more plan-based and analysis-driven activities in software startups that invest in their workflow. Mansoori and Lackeus (2019) propose a framework for applying entrepreneurial methods consisting of the three levels of logic, model, and tactics. While we have not seen the consistent appearance of causation and effectuation at the model level, we observe that they are relevant at the tactical level.

Tactics connect the abstract nature of the logics to the tangible realm of practices. Tactics are often detailed and specify the context of use and the outcomes of action. Several engineering phenomena, such as MVP, TD, lead users, and test-driven development, can be described using causation or effectuation logic at the tactic level. Reymen et al. (2017) studied decision-making logics in four high-tech startups. The authors found both effectual and causal logics in different parts of startups' business models. In our own study, we see the appearance of both logics in different parts of the product development lifecycle. Harms and Schiele (2012) found that, for business development, the entrepreneurs' experiences might also influence the choice of entrepreneurial logic, not only the



surrounding environments. Dew et al. (2009) also found that entrepreneurial experts frame decisions using effectual logic while novices use a causation-driven approach and tend “to go by the textbook.” (Dew et al. 2009, p. 1) It might also be that the dominant decision-making logic may shift several times (Reymen et al., 2015), and both decision-making logics may co-exist according to the different degrees of uncertainty in the market and technology or the number of decision-makers involved (Nummela et al., 2014). We observed some contextual variables that might determine whether an effectual mindset or a causal mindset should be in place, including business uncertainty, startup maturity, funding situations, and expertise in engineering methods regarding SE activities. Future work can further investigate these as factors that precede the choice of logic in software startup engineering.

Existing research has also revealed that it is possible to observe both kinds of entrepreneurial logic in different stages of a startup. Smolka et al. reported that causation and effectuation both have positive effects on business development (Smolka et al., 2016). Founders who are resource-driven and engage in planning activities tend to have better startup performance. This is a fascinating observation; however, we did not have enough insight to validate this combined effect in software development contexts. The logic shifts may also explain the changes in startups’ business strategies, marketing approaches, and workflows (Reymen et al., 2017; Sarasvathy, 2001; Harms and Schiele, 2012). Looking at startups in different stages, we hypothesize that startup founders can shift their logic from effectuation-driven towards causation-driven by gradually establishing their workflow in different SE activities.

### 5.3. Threats to Validity

In qualitative research, scientific validity must be addressed to replicate research and ensure that the findings are trustworthy (Yin, 2003; Runeson and Höst, 2009; Cruzes and Dyba, 2011). To ensure the validity of this study, we followed the validity guidelines from Runeson (Runeson and Höst, 2009). Construct validity ensures that the studied operational variables represent the construct we aim to investigate according to the research questions. In our study, the components were developed based on existing literature (McKelvie et al., 2020). This study’s measure of entrepreneurial logic is based on approaches reported in previous studies (Reymen et al., 2015; McKelvie et al., 2020). Our interview questions reveal major key events in each startup, reflected by CEOs or startup co-founders with insights into business and product aspects. A possible risk here is bias in data, i.e. interviewee might focus on the most improvised or messy aspects of the product development. We implemented some measures against this threat. First, we read again the interview transcripts, which captured also emotional expressions to recall how the interviewee presented themselves. Second, we investigated the context of main labeled events to detect if there is any visible bias. It is not easy to understand a startup and its cultural, institutional, and contextual factors within a single interview of about 60 minutes; therefore, we compensated by collecting data about the startups through incubator and company websites before interviews. We also talked to co-authors or researchers (if available) who have connections to the case to better understand the socio-cultural context of the case.

To improve the study’s reliability, we invited all participating startups to proofread the (part of) results to ensure their conformance with reality. Moreover, we had several rounds of discussions after data analysis among authors to allow alternative interpretations and regulate possible over-interpretations. The review process from the *Empirical Software Engineering* journal also helps us critically reviewing initial research questions and make additional adjustments and analyses.

Internal validity concerns causal relations between investigating factors, such as our entrepreneurial logic, and making engineering decisions. Our study explores the logic’s occurrence across types of SE activities and startup cases and does not aim to associate a relationship. Therefore, this particular limitation is not a concern in this study.

External validity refers to the extent to which the findings are generalizable beyond the context studied. For qualitative studies, the intention is to enable analytical generalization where the results are extended to companies with common characteristics. We have tried in different ways to achieve the diversity and representativeness of our cases. However, it was very difficult to reach out and talk to startups. We did not detect any systematic bias due to the possible differences between the ones who accepted and the ones who refused to participate. By emailing and talking via phone, professional networks and nearby startups have a slightly lower turnover rate than startups from Crunchbase. However, when approaching startups via personal introduction or meet in person, there is a significantly higher chance to acquire their participations.

Our case sample is skewed toward specific geographical locations (Norway and Nordic countries in general), startup phases (dominated by companies in either pre-startup or startup stages), team sizes (mostly between three to 20 people), and funding model (mostly by bootstrapping). Consequently, it would be safe to relate these findings to startups with similar characteristics (i.e., European software startups). Startups from other American countries or startups already in a growth stage might not share the features observed in the majority of our cases; they may be more causal than effectual, for example. Another remark is that our findings apply to the startups at the time at which they were investigated. The study was not designed as a longitudinal case study; hence, we do not claim that entrepreneurial logic will appear in the same way in these startups at another point in time.

Reliability refers to the extent to which data and the analysis are dependent on the specific researchers. We have defined and validated interview protocols with colleagues. Some interviews were in Norwegian. We tried our best to preserve the actual meaning of respondents via the transcription. Recordings were transcribed shortly after each interview to mitigate bias. We have cross-checked the analysis results between the first and second authors of this study, and a high consensus level was reached. Additionally, we compared findings to related literature (Giardino et al., 2016; Hevner and Malgonde, 2019; Klotins et al., 2019; Melegati et al., 2019), examining similarities, contrasts, and explanations. Such comparisons have enhanced the internal validity and quality of our findings (Eisenhart, 1989).

## 6. Conclusions

As software startups find themselves operating in uncertain, risky, and dynamic environments, existing software development approaches have limited applicabilities due to their prediction-based theoretical underpinnings. Our goal is to increase current knowledge about SE in startup contexts by adopting the entrepreneurial logic lens. From a qualitative survey of 40 startups, we observe dominant effectuation-driven software development behaviors that focus on requirement engineering, software construction, process management, software design, and maintenance. Effectuation-driven approaches promise to develop different processes, models, and tactics that welcome uncertainty and risk. We showed that both entrepreneurial logics occur and help advance the current understanding of the *how* and *why* of engineering processes and practices in startups. For instance, TD acceptance, requirement identification, and MVPs tend to be driven by effectual logics, while causal logic drives test-driven development.

Future research will explore how best to build software development methods that incorporate aspects of entrepreneurial behavior logic. We propose three potential areas of future research:

- Making the right decisions is essential for entrepreneurial application success in software startups. The effectuation-driven approach to software development supports a new way to take actions that fit existing means but still consider the long-term goals.
- We need to better understand the influence of entrepreneurial contexts on the occurrence of behavior logics. We do not have enough data to compare and analyze different environment conditions and relate them to the frequency of entrepreneurial logics across SE activities.
- Logic can shift from effectuation-driven to causation-driven software development. We had mostly cross-sectional views into startup cases, which limited our observation of the behavior logics' temporal evolution. Founders and managers need to understand how and under which conditions the effectuation-driven behaviors change to causation-driven ones.

## References

- Alvarez C (2014): *Lean Customer Development*, O'Reilly Media, Sebastopol, CA.
- Alvarez SA, Barney JB (2005) How do entrepreneurs organize firms under conditions of uncertainty? <https://doi.org/10.1177/0149206305279486>
- Aurum A, Wohlin C (2003) The fundamental nature of requirements engineering activities as a decision-making process. *IST 45*(14): 945–954 <http://www.sciencedirect.com/science/article/pii/S095058490300096X>
- Ayala C, Nguyen-Duc A, Franch X, Höst M, Conradi R, Cruzes D, Babar MA (2018) System requirements-OSS components: matching and mismatch resolution practices – an empirical study. *Empir Softw Eng 23*(6): 3073–3128. <https://doi.org/10.1007/s10664-017-9594-1>
- Bajwa SS (2020) Pivoting in software startups. In: Nguyen-Duc A, Münch J, Prikladnicki R, Wang X, Abrahamsson P (eds) *Fundamentals of software startups: Essential engineering and business aspects*. Springer International Publishing, Springer Nature Switzerland AG, pp 27–43
- Bajwa SS, Wang X, Duc AN, Abrahamsson P (2017) “Failures” to be celebrated: an analysis of major pivots of software startups. *Empir Softw Eng 22*(5):2373–2408
- Barney J (1991) Firm resources and sustained competitive advantage. *J Manage 17*(1): 99–120 <https://doi.org/10.1177/014920639101700108>
- Barton Cunningham, J., Gerrard, P., Schoch, H., & Lai Hong, C. (2002). An entrepreneurial logic for the new economy. *Management Decision, 40*(8), 734–744. <https://doi.org/10.1108/00251740210437707>
- Beck K, Andres C (2004): *Extreme Programming Explained: Embrace Change*, 2nd edn. Addison-Wesley Professional, Boston
- Beck K, Beedle M, van Bennekum A, Cockburn A, Cunningham W, Fowler M, et al. Manifesto for Agile Software Development [Internet]. Manifesto for Agile Software Development. 2001. Available from: <http://www.agilemanifesto.org/>
- Berends H, Jelinek M, Reymen I, Stultiëns R (2013) Product innovation processes in small firms: Combining entrepreneurial effectuation and managerial causation. *J. Prod. Innov. 31*(3):616–635. <https://onlinelibrary.wiley.com/doi/abs/10.1111/jpim.12117>
- Berg V, Birkeland J, Nguyen-Duc A, Pappas I, Jaccheri L (2018) Software startup engineering: A systematic mapping study. *J Syst Softw 144*: 255-274
- Berg V, Birkeland J, Nguyen-Duc A, Pappas IO, Jaccheri L (2020) Achieving agility and quality in product development: an empirical study of hardware startups. *J Syst Softw. 167*, <https://doi.org/10.1016/j.jss.2020.110599>
- Blank S (2013) An MVP is not a cheaper product it's about smart learning. Steve Blank. URL: <https://steveblank.com/2013/07/22/an-mvp-is-not-a-cheaper-product-its-about-smart-learning/> Access dated: April 2019
- Blank SG (2007): *The Four Steps to the Epiphany*, 2nd ed., Cafepress.com, San Francisco.
- Blank SG and Dorf B (2012): *The Startup Owner's Manual: The Step-By-Step Guide for Building a Great Company*, K&S Ranch, Pescadero, CA.

- Boland RJ (2008) Decision making and sensemaking. In: Burstein F, Holsapple CW (eds) Handbook on decision support systems 1: Basic themes international handbooks information system. Springer-Verlag Berlin Heidelberg, pp 55–63
- Bosch J, Holmström Olsson H, Björk J, Ljungblad J. (2013): The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups. In: Fitzgerald B, Conboy K, Power K, Valerdi R, Morgan L, Stol K-J, editors. Lean Enterprise Software and Systems. Berlin, Heidelberg: Springer; 2013. pp 1–15
- Bourque P, Fairley RE (2014) Guide to the software engineering body of knowledge (SWEBOK (R)): Version 30. IEEE Computer Society Press
- Boyatzis RE (1998) Transforming qualitative information: Thematic analysis and code development. Sage Publications Inc
- Braun V, Clarke V (2006) Using thematic analysis in psychology. *Qual Res Psychol* 3(2):77–101. <https://doi.org/10.1191/1478088706qp063oa>
- Brettel M, Mauer R, Engelen A, Küpper D (YEAR): Corporate effectuation: Entrepreneurial action and its impact on R&D project performance. *J Bus Ventur* 27(2):167–184. <https://doi.org/10.1016/j.jbusvent201101001>
- Bygrave WD, Hofer CW (1991) Theorizing about entrepreneurship. *Entrepreneurship: Theory and Practice* 16(2):13–22. <https://doi.org/10.1177/104225879201600203>
- Chandler GN, DeTienne DR, McKelvie A, Mumford TV, (2011) Causation and effectuation processes: A validation study. *J Bus Ventur* 26(3):375–390. <https://doi.org/10.1016/j.jbusvent200910006>
- Cico O, Duc AN, Jaccheri L. (2020): An Empirical Investigation on Software Practices in Growth Phase Startups. In: Proceedings of the Evaluation and Assessment in Software Engineering [Internet]. New York, NY, USA: Association for Computing Machinery; 2020. pp.282–7
- Collis J, Hussey R (2009) Business research: A practical guide for undergraduate and postgraduate students. Palgrave MacMillan, UK. <https://www.macmillanihocom/page/detail/Business-Research/?K=9780230301832>
- Corbin JM, Strauss A (1990) Grounded theory research: Procedures canons and evaluative criteria. *Qual Sociol* 13(1):3–21. <https://doi.org/10.1007/BF00988593>
- Cruzes DS, Dyba T (2011) Recommended steps for thematic synthesis in software engineering In: 2011 International symposium on empirical software engineering and measurement. IEEE, pp 275–284. <https://doi.org/10.1109/ESEM2011.36>
- Daher M, Carré D, Jaramillo A, Olivares H, Tomicic A. (2017): Experience and Meaning in Qualitative Research: A Conceptual Review and a Methodological Device Proposal. *FQS* 2017;18(3).
- de O Melo C S, Cruzes D, Kon F, Conradi R (2013) Interpretative case studies on agile team productivity and management. *Inf Softw Technol* 55(2):412–427. <https://doi.org/10.1016/j.infsof201209004>
- Dew N, Read S, Sarasvathy SD, Wiltbank R (2009) Effectual versus predictive logics in entrepreneurial decision-making: Differences between experts and novices. *J Bus Ventur* 24(4):287–309. <https://doi.org/10.1016/j.jbusvent200802002>
- Duc AN, Abrahamsson P (2016) Minimum viable product or multiple facet product? The role of MVP in software startups In: Sharp H, Hall T (eds) Agile processes in software engineering and extreme programming lecture notes in business information processing. Springer International Publishing, Cham, pp 118–130

Easterbrook S, Singer J, Storey MA, Damian D (2008) Selecting empirical methods for software engineering research. Springer, London, pp 285–311

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting Empirical Methods for Software Engineering Research. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to Advanced Empirical Software Engineering* (pp. 285–311). Springer. [https://doi.org/10.1007/978-1-84800-044-5\\_11](https://doi.org/10.1007/978-1-84800-044-5_11)

Eisenhardt KM (1989) Building theories from case study research. *Acad Manage Rev* 14(4):532–550

Fagerholm F, Sanchez Guinea A, Mäenpää H, Münch J (2017) The right model for continuous experimentation. *J Syst Soft* 123:292–305. <https://doi.org/10.1016/j.jss.201603034>

Fisher G (2012): Effectuation causation and bricolage: A behavioral comparison of emerging theories in entrepreneurship research. *ETP* 36(5):1019–1051

Gautam N and Singh N (2008): Lean product development: Maximizing the customer perceived value through design change (redesign). *International Journal of Production Economics*, vol. 114(1): pp. 313–332.

Gemmell RM, Boland RJ, & Kolb DA (2012): The socio-cognitive dynamics of entrepreneurial ideation. *Entrepreneurship Theory & Practice*, 36(5):1053–1073

Ghezzi A (2018): Digital startups and the adoption and implementation of lean startup approaches: Effectuation bricolage and opportunity creation in practice. <https://doi.org/10.1016/j.techfore.201809017>

Giardino C, Bajwa SS, Wang X, Abrahamsson P (2015) Key challenges in early-stage software startups. In: Lassenius C, Dingsøyr T, Paasivaara M (eds) *Agile processes in software engineering and extreme programming lecture notes in business information processing*. Springer International Publishing, Cham, pp 52–63

Giardino C, Paternoster N, Unterkalmsteiner M, Gorschek T, Abrahamsson P (2016): Software development in startup companies: The greenfield startup model. *IEEE Trans Softw Eng* 42(6):585–604. <https://doi.org/10.1109/TSE.2015.2509970>

Giardino C, Unterkalmsteiner M, Paternoster N, Gorschek T, Abrahamsson P. What Do We Know about Software Development in Startups? *IEEE Software*. 2014 Sep;31(5):28–32.

Giardino C, Wang X, Abrahamsson P (2014) Why early-stage software startups fail: a behavioral framework In: *International conference of software business*. Springer, Cham, pp 27–41

Gothelf J (2013): *Lean UX: Applying Lean Principles to Improve User Experience*. USA: O'Reilly

Harms R, Schiele H (2012) Antecedents and consequences of effectuation and causation in the international new venture creation process. *IJE* 10(2):95–116. <https://doi.org/10.1007/s10843-012-0089-2>

Hevner A, Malgonde O (2019) Effectual application development on digital platforms. *Electron Mark* 29(3):407–421 <https://doi.org/10.1007/s12525-019-00334-1>

Highsmith J, Cockburn A (2001): *Agile software development: the business of innovation*. *Computer* 34(9):120–127

Hokkanen L, Väänänen-Vainio-Mattila K (2015): UX Work in Startups: Current Practices and Future Needs. In: Lassenius C, Dingsøyr T, Paasivaara M, editors. *Agile Processes in Software Engineering and Extreme Programming*. Cham: Springer International Publishing; 2015. p. 81–92

- Kemell KK, Ventilä E, Kettunen P, Mikkonen T (2019) Amidst uncertainty – or not? Decision-making in early-stage software startups. In: Hyrynsalmi S, Suoranta M, Nguyen-Duc A, Tyrvaïnen P, Abrahamsson P (eds) *Software business lecture notes in business information processing*. Springer, Cham, pp 369–377
- Khanna D, Nguyen-Duc A, Wang X (2018) From MVPs to pivots: A hypothesis-driven journey of two software startups. In: Wnuk K, Brinkkemper S (eds) *Software business lecture notes in business information processing*. Springer International Publishing, Cham, pp 172–186
- Khurum M, Fricker S, Gorschek T (2015): The contextual nature of innovation – an empirical investigation of three software intensive products. *Inf Softw Technol* 57:595–613. <https://doi.org/101016/jinfsof201406010>
- Klotins E, Unterkalmsteiner M, Chatzipetrou P, Gorschek T, Prikladniki R, Tripathi N, Pompermaier L (2019) A progression model of software engineering goals challenges and practices in start-ups. *IEEE Trans Softw Eng*, <https://doi.org/101109/TSE20192900213>
- Klotins E, Unterkalmsteiner M, Gorschek T (2019): Software engineering in start-up companies: An analysis of 88 experience reports. *Empir Softw Eng* 24(1):68–102. <https://doi.org101007/s10664-018-9620-y>
- Kruchten P, Nord RL, Ozkaya I (2012) Technical debt: From metaphor to theory and practice. *IEEE Softw* 29(6):18–21
- Kraaijenbrink J (2008) The nature of the entrepreneurial process: Causation effectuation and pragmatism. In: Groen A, Oakey R, Van Der Sijde P, Cook G (eds) *New technology-based firms in the new millennium* 9. Emerald Group Publishing Limited, Bingley, pp 187. [https://doi.org/101108/S1876-0228\(2012\)0000009015](https://doi.org/101108/S1876-0228(2012)0000009015)
- Ladd T, Lyytinen K, & Gemmell R (2015): How customer interaction and experimentation advance new venture concepts in a cleantech accelerator. *Academy of Management Proceedings*, 2015(1), 11415.
- Lindgren E, Münch J (2016) Raising the odds of success: the current state of experimentation in product development. *IST* 77:80–91. <https://doi.org/101016/jinfsof201604008>
- Mansoori Y, Lackéus M (2019) Comparing effectuation to discovery-driven planning prescriptive entrepreneurship business planning lean startup and design thinking. *Small Bus Econ* 54(3):791–818. <https://doi.org/101007/s11187-019-00153-w>
- Mansoori, Y. (2015). *Entrepreneurial methods*. Licentiate dissertation. Gothenburg, Sweden: Chalmers University of Technology.
- Mansoori, Y., & Lackéus, M. (2020). Comparing effectuation to discovery-driven planning, prescriptive entrepreneurship, business planning, lean startup, and design thinking. *Small Business Economics*, 54(3), 791–818. <https://doi.org/10.1007/s11187-019-00153-w>
- McKelvie A, Chandler GN, DeTienne DR, Johansson A (2020) The measurement of effectuation: highlighting research tensions and opportunities for the future. *Small Bus Econ* 54(3):689–720. <https://doi.org/101007/s11187-019-00149-6>
- Melegati J, Goldman A, Kon F, Wang X (2019) A model of requirements engineering in software startups. *IST* 109: 92–107. <https://doi.org/101016/jinfsof201902001>
- Nambisan S (2017) Digital entrepreneurship: Toward a digital technology perspective of entrepreneurship. *ETP* 41(6):1029–1055. <https://doi.org/101111/etap12254>
- Nguyen-Duc A, Münch J, Prikladnicki R, Wang X, Abrahamsson P (eds) (2020) *Fundamentals of software startups: Essential engineering and business aspects*, 1st edn. Springer Nature Switzerland AG.

- Nguyen-Duc A, Dahle Y, Steinert M, Abrahamsson P (2017) Towards understanding startup product development as effectual entrepreneurial behaviors. In: Felderer M, Méndez Fernandez D, Turhan B, Kalinowski M, Sarro F, Winkler D (eds) *Product-focused software process improvement lecture notes in computer science*. Springer International Publishing, Cham, pp 265–279
- Nguyen-Duc A, Seppanen P, Abrahamsson P (2015) Hunter-gatherer cycle: A conceptual model of the evolution of software startups. In: *ICSSP 2015: Proceedings of the 2015 International Conference on Software and System Process*. ACM, New York, NY, USA, pp 199–203. <https://doi.org/10.1145/27855922795368>
- Nguyen-Duc A, Shah SMA, Amrahamsson P (2016): Towards an early-stage software startups evolution model. In: *Software Engineering and Advanced Applications (SEAA) 2016: 42nd Euromicro Conference*. IEEE, pp 120–127
- Nguyen-Duc A, Wang X, Abrahamsson P (2017) What influences the speed of prototyping? An empirical investigation of twenty software startups. In: Baumeister H, Lichter H, Reinisch M (eds) *Agile processes in software engineering and extreme programming lecture notes in business information processing*. Springer International Publishing, pp 20–36
- Norman D and Draper S (1986): *User Centered System Design: New Perspectives on Human-Computer Interaction*. FL, USA: CRC Press, 1986.
- Nummela N, Saarenketo S, Jokela P, Loane S (2014) Strategic decision-making of a born global: A comparative study from three small open economies. *MIR* 54(4):527–550. <https://doi.org/10.1007/s11575-014-0211-x>
- Oates BJ (2005) *Researching information systems and computing*. Sage, Newbury Park, California
- Ojala A (2015): Business models and opportunity creation: How IT entrepreneurs create and develop business models under uncertainty 26(5) 451–476 DOI 10.1111/isj12078
- Ojala A (2016) Discovering and creating business opportunities for cloud services. *J Syst Softw* 113:408–417
- Pantiuchina J, Mondini M, Khanna D, Wang X, Abrahamsson P (2017) Are software startups applying agile practices? The state of the practice from a large survey. In: Baumeister H, Lichter H, Reinisch M (eds) *Agile processes in software engineering and extreme programming lecture notes in business information processing*. Springer, Cham, pp 167–183
- Paternoster N, Giardino C, Unterkalmsteiner M, Gorschek T, Abrahamsson P (2014) Software development in startup companies: A systematic mapping study. *Inf Softw Technol* 56(10):1200–18. <https://doi.org/10.1016/j.infsof.2014.04.014>
- Pitchbook (2019) 4q 2018 PitchBook-NVCA venture monitor Pitchbook <https://pitchbook.com/news/reports/4Q-2018-pitchbook-nvca-venture-monitor> Date assessed: April 2019
- Ralph P (2016) Software engineering process theory: A multi-method comparison of sensemaking-coevolution-implementation theory and function-behavior-structure theory. *IST* 70:232–250. <https://doi.org/10.1016/j.infsof.2015.06.010>
- Read S, Song M, Smit W (2009) A meta-analytic review of effectuation and venture performance. *J Bus Ventur* 24(6):573–587. <https://doi.org/10.1016/j.jbusvent.2008.02.005>
- Reymen I, Andries P, Berends H, Mauer R, Stephan U, Burg EV (2015) Understanding dynamics of strategic decision making in venture creation: A process study of effectuation and causation. *Strateg Entrepreneurship J* 9(4):351–379. <https://doi.org/10.1002/sej1201>

Reymen I, Berends H, Oudehand R, Stultiëns R (2017) Decision making for business model development: a process study of effectuation and causation in new technology-based ventures. *R D Manag.* 47(4):595–606. <https://doi.org/10.1111/radm.12249>

Ries E (2011) *The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses.* Crown Publishing Group, USA

Runeson P, Höst M (2009): Guidelines for conducting and reporting case study research in software engineering. *Empir Softw Eng* 14(2): 131

Sarasvathy SD (2001) Causation and effectuation: Toward a theoretical shift from economic inevitability to entrepreneurial contingency. *Acad Manage Rev* 26(2):243–263

Sarasvathy SD, Dew N (2005a) Entrepreneurial logics for a technology of foolishness. *SJM* 21(4):385–406. <https://doi.org/10.1016/j.jscaman.2005.09.009>

Sarasvathy SD, Dew N (2005b) New market creation through transformation. *J Evol Econ* 15(5):533–565. <https://doi.org/10.1007/s00191-005-0264-x>

Seaman C, Guo Y (2011) Measuring and monitoring technical debt. *Adv Comput* 82:25–46

Seaman CB (1999) Qualitative methods in empirical studies of software engineering. *IEEE Trans Softw Eng* 25(4):557–572

Signoretti I, Marczak S, Salerno L, Lara A d, Bastos R. (2019): Boosting Agile by Using User-Centered Design and Lean Startup: A Case Study of the Adoption of the Combined Approach in Software Development. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–6.

Smolka KM, Verheul I, Burmeister-Lamp K, Heugens PP (2016) Get it together! synergistic effects of causal and effectual decision-making logics on venture performance. *ETP* 42(4): 571-604 <https://doi.org/10.1111/etap.12266>

Steininger DM (2019) Linking information systems and entrepreneurship: A review and agenda for IT-associated and digital entrepreneurship research. *Inf Syst J* 29(2):363–407. <https://doi.org/10.1111/isj.12206>

Stol, K., Ralph, P., & Fitzgerald, B. (2016). Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 120–131. <https://doi.org/10.1145/2884781.2884833>

Sutton SM (2000) The role of process in software start-up. *IEEE Soft* 17(4):33–39. <https://doi.org/10.1109/52854066>

Tell P, Klünder J, Küpper S, Raffo D, MacDonell SG, Münch J, et al. (2019): What are Hybrid Development Methods Made Of? An Evidence-Based Characterization. In: 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP). 2019. pp. 105–14.

Tripathi N, Klotins E, Prikladnicki R, Oivo M, Pompermaier LB, Kudakacheril AS, Unterkalmsteiner M, Liukkunen K, Gorschek T (2018) An anatomy of requirements engineering in software startups using multi-vocal literature and case survey. *J Syst Soft* 146:130–151. <https://doi.org/10.1016/j.jss.2018.08.059>

Tripathi N, Oivo M, Liukkunen K, Markkula J (2019) Startup ecosystem effect on minimum viable product development in software startups. *IST* 114:77–91

Trochim, W. (2001). *Research Methods Knowledge Base.* Atomic Dog Publishing, Cincinnati, OH, USA.



Tuckman BW (1965): Developmental sequence in small groups. *Psychol Bull* 63(6):384–399.

Unterkalmsteiner M, Abrahamsson P, Wang XF, Anh ND, Shah S, Bajwa SS, Baltés GH, Conboy K, Cullina E, Dennehy D, Edison H, Fernandez-Sanchez C, Garbajosa J, Gorschek T, Klotins E, Hokkanen L, Kon F, Lunesu I, Marchesi M, Morgan L, Oivo M, Selig C, Seppanen P, Sweetman R, Tyrvaïnen P, Ungerer C, Yague A (2016) Software startups: a research agenda. *E-Informatica Software Engineering Journal* 10(1):89–123. <https://doi.org/105277/e-Inf160105>

von Hippel E (1986) Lead users: A source of novel product concepts. *Manage Sci* 32(7):791–805. <https://doi.org/101287/mnsc327791>

Walsham G (1995): The emergence of interpretivism in IS research. *Inf Syst Res* 6(4):376–394. <https://www.jstor.org/stable/23010981>

Weick, K. (1995). *Sensemaking in Organisations*. London: Sage.

Weick, K., Sutcliffe, K. M., & Obstfeld, D. (2005). Organizing and the process of sensemaking. *Organization Science*, 16(4): 409–421

Wiltbank R, Dew N, Read S, Sarasvathy SD (2006) What to do next? The case for non-predictive strategy 27(10):981–998. <https://doi.org/101002/smj555>

Wohlin C, Aurum A (2015): Towards a decision-making structure for selecting a research design in empirical software engineering. *Empir Softw Eng* 20(6):1427–1455. <https://doi.org/101007/s10664-014-9319-7>

Yaman SG, Sauvola T, Riungu-Kalliosaari L, Hokkanen L, Kuvaja P, Oivo M, Männistö T (2016) Customer involvement in continuous deployment: A systematic literature review. In: Daneva M, Pastor O (eds) *Requirements engineering: Foundation for software quality lecture notes in computer science*. Springer International Publishing, Cham, pp 249–265.

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



V

**A CARD-BASED METHOD FOR EARLY-STAGE SOFTWARE  
STARTUPS**

by

Kai-Kristian Kemell, Anh Nguyen-Duc, Mari Suoranta & Pekka Abrahamsson,  
2022

*Submitted to a journal for review in January 2022. Being revised for the second  
review round at the time of this dissertation's publication*

Request a copy from author.