

Tuomas Taubert

MODERN MOBILE APPLICATION DEVELOPMENT IN A STARTUP

ABSTRACT

Tuomas Taubert: Modern Mobile Application Development in a Startup
M.Sc. Thesis
Tampere University
Master's Degree Programme in Software Development
July 2023

Mobile applications are an everyday part of our modern life, and oftentimes new mobile applications are developed by startups. Usually, startups have acutely limited resources to work with, so they need to work innovatively and fast in the difficult early stages of their journey. While the technologies and the software development methods utilized in the development of mobile applications have both been extensively studied, not as much research has been done about these topics in the startup context.

Therefore, this thesis concentrated on two major themes: the different mobile application technologies available for startups and different software development methods that startups can utilize in the development process. First, a literature review of these topics was conducted, after which the theoretical background was used to perform an action research study with a startup. During the action research, the startup developed a new version of their application with a new technology Flutter and explored different software development methods.

The results showed that a cross-platform solution like Flutter worked well for the startup in question. In addition to the cross-platform functionality enabling a single codebase, the code formatting tools and the style guide included were practical for keeping the codebase consistent and maintainable. Lastly, the developer tools and the widget-based coding style helped in code duplication and development pace, which were previously problematic for the startup. The development method was first changed from a free-form development to Kanban and later to Scrum, which was found to be the most suitable solution for this team. For each of the development methods, the startup team made changes to the model to better suit their situation, which appears to be common among startups in the literature as well.

Keywords: Startup, mobile application, agile methods, cross-platform, action research.

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

Contents

1	Introduction.....	1
1.1	Research questions and methods	2
1.2	Literature review	2
1.3	Structure of the thesis	3
2	Modern mobile applications.....	4
2.1	Classifying different mobile application approaches	4
2.2	Native applications	5
2.3	Web and hybrid applications	5
2.4	Interpreted applications	7
2.5	Cross-compiled applications	8
3	Software development practices	10
3.1	Traditional software development practices	10
3.2	Agile methods	12
3.2.1	Scrum	13
3.2.2	Extreme Programming	16
3.2.3	Lean	18
3.2.4	Kanban	20
4	Developing mobile applications in a startup	24
4.1	Feasibility of different technical approaches for startups	26
4.2	Software development practices in a startup	27
5	Action research on developing LessonApp.....	33
5.1	Research methodology	33
5.2	About LessonApp	35
5.3	The current state of development	36
5.4	Conducting the action research: first cycle	37
5.4.1	Diagnosis	37
5.4.2	Plan	37
5.4.3	Act	38

5.4.4 Evaluate	39
5.4.5 Reflect	40
5.5 Conducting the action research: second cycle	41
5.5.1 Diagnosis	41
5.5.2 Plan	42
5.5.3 Act	42
5.5.4 Evaluate	43
5.5.5 Reflect	45
5.6 End of the second cycle	47
6 Results	47
6.1 Modern mobile application development frameworks for startups	47
6.2 The best practices for the software development process in startups	49
6.3 The best practices in practice in a startup	51
7 Conclusions	54
7.1 Limitations	56
7.2 Future work	57
References	59

1 Introduction

Modern problems require modern solutions, and increasingly often, new mobile applications are developed to address them. Oftentimes these solutions are created by software startups that are vying to find a place in the market for their product. Usually, startups have acutely limited resources to work with, so they need to work innovatively and fast in the difficult early stages of their journey. This fever-paced development often means that startups forgo more formal development processes while still following many of the agile principles in their work. These limitations and requirements make startups interesting research subjects from the perspective of software development research, as such a company model poses new kinds of challenges to the software development process.

Traditional mobile application development would often be too resource-intensive, time-consuming, and expensive to be feasible for startups, so faster and more flexible development processes must be adopted. The technological aspect should be examined as well: what kind of solutions can provide fast-paced and economically viable development options for startups?

The motivation for this thesis came from the author's work in a startup in the field of pedagogy specializing in lesson planning. The realities of the work in a startup often did not line up with expectations, and the lack of clear process planning and management made the development work feel erratic and unfocused. This observation led to the conclusion that the development process could most likely be made into a much more efficient and effective one. Studying the situation in the startup and the effects of the changes they would decide to make could yield some findings that could help both existing and upcoming startups, as well as contribute to the scientific community. These possibilities made it worthwhile to research this topic in this thesis.

While startups should be an interesting research target for academia as well, many of the slightly older studies note a general lack of research about startups, and only a few studies have been done into the software development practices in startups [Sutton 2000, Giardino et al. 2016, Abrantes and Furtado 2021]. A more recent survey of the literature by Carroll and others [2022] noted that the number of primary studies into software startups had been rapidly growing until the COVID-19 pandemic, which shut down many

universities and research institutions worldwide. Still, compared to the numerous different research areas identified in the same study, the number of studies published remains relatively low. This serves to highlight the importance of conducting more primary research into startups and their software development practices.

1.1 Research questions and methods

To best explore the topic of software development in startups, a combination of theoretical background and practical exploration was deemed to be the most effective approach. With the author's own work in a startup, it was possible to conduct an action research study into the work of this particular startup. While there are many different ways in which one could approach this topic, this thesis concentrates on two major themes: the different mobile application technologies available for startups and different software development methods that can be utilized in the development process. These themes were selected based on a preliminary analysis of the current situation of the startup and the areas where the action research study could yield the most benefits to both the research and the startup itself.

To understand the context for this thesis, it's essential to first conduct a literature review about the topic. The knowledge gained from this is then used in the action research study in the startup. Therefore, this thesis first examines the literature for the most suitable practices for developing a mobile application in a startup context and then explores these practices in real life context during the action research. This led to the following research questions:

1. What modern mobile application development frameworks are available and suitable for startups?
2. What are the best practices for the software development process in startups found in the literature?
3. How well do the best practices work in practice for a startup?

The first and second research questions will be examined with the literature review, while the action research will be used to investigate the third research question.

1.2 Literature review

The main channels used to search for relevant literature were the Tampere University Library's Andor service and Clarivate's Web of Science. Andor was used to search for

relevant articles and other literature, while the Web of Science was also used to browse the citation network of the most relevant articles for other literature that might not have been found in the searches in Andor. Search queries were made with combinations of the following keywords, utilizing the AND and OR operators to combine terms for search queries:

1. Technologies: “cross-platform”, “hybrid”, “mobile”, “app” / “application”.
2. Software development methods: “software development”, “Agile”, “Lean”, “methods”. The search was continued with the most relevant terms: “Scrum”, “Extreme Programming”, “Kanban”.
3. “startup” was a keyword that was combined with the previous searches to also get startup-specific results.

From the results of these searches, articles and other sources were picked based on their relevance and applicability to the thesis topic, concentrating on sources discussing startups specifically. Most of the selected sources were from peer-reviewed publications, but some other sources describing more general practices were used as well while taking into consideration their accuracy and trustworthiness.

1.3 Structure of the thesis

The research questions and their background are discussed in the following chapters. Chapter 2 focuses on the state of the technological side of mobile application development. The most common software development practices are then discussed in Chapter 3, and both of these are then examined from the startup point of view in Chapter 4. Afterward, the theoretical background is used to conduct a study in a startup context in Chapter 5, and the results will be evaluated in Chapter 6 of the thesis. Finally, Chapter 7 concludes the thesis.

2 Modern mobile applications

The mobile application marketplace is nowadays dominated by two big platforms: Apple's iOS and Google's Android. At the time of writing, their market share is approaching over 99% of mobile devices accessing websites on the Internet [Statcounter 2021]. Unfortunately for startups, developing an application for one of these platforms does not automatically mean that the application can be used in the other without modifications. In the worst case, the startup would need to develop two completely separate applications and keep maintaining them on the two different platforms, and this is understandably something the startups would want to avoid if possible. To address this problem, many different approaches have been developed to facilitate cross-platform application development.

2.1 Classifying different mobile application approaches

The traditional classification of mobile applications has been a division into three categories: *native*, *web*, and *hybrid* applications [Jakopec and Vilček 2017]. This division is based on the idea that the applications can be developed either on the platform's own specific development environment with the platform's "own" coding language (*native*), web technologies (*web*), or some other approach that enables cross-platform development (*hybrid*) [Jakopec and Vilček 2017]. However, this categorization has been challenged in some of the more recent research, especially because of some solutions that blur the line between the web and hybrid applications and also because of the very diverse solutions in the hybrid application category [Nunkesser 2018].

Instead of the traditional classification, Nunkesser [2018] has proposed a division into *endemic*, *pandemic*, and *ecdemic* applications, which roughly correspond to *native*, *web/hybrid*, and *interpreted or generated hybrid* applications. These categories are then further divided into subcategories. However, these terms or other alternative ones have not been adopted into wider use, and the author himself comments that the traditional classification has been the standard for a long time now, so it might not even be possible to replace them.

One useful categorization for cross-platform applications was defined by Raj and Tolety [2012], where the different approaches were divided into four categories: *web*, *hybrid*, *interpreted*, and *cross-compiled* applications. In this categorization, the two extra

categories have been added to better differentiate the versatile solutions. For the needs of this thesis, this four-part categorization of cross-platform solutions is sufficient.

2.2 Native applications

Both Android and iOS require the use of a particular programming language, different development environments, and platform-specific APIs when developing native applications on their respective platform. The native applications are programmed in Java or Kotlin for Android, or Objective-C or Swift for iOS. This means that if a company wanted to provide a native application on both platforms, it would need to develop and maintain two different versions of its product. While the platform-specific native applications should be able to effectively use the potential of their platform, the difficulty and overhead of maintaining different versions of the application increase development time and maintenance costs for the company developing the application [Xanthopoulos and Xingalos 2013].

It's difficult for cross-platform solutions to outperform the native applications in most performance measurements, such as smaller package size or the speed of startup or execution, except in some more special cases. This can be seen in the works of e.g., Kaczmarczyk and Wojciech Zabierowski [2021], Saarinen [2019], and Ferreira and others [2018]. Despite this, there are also some studies showing cross-platform solutions sometimes having better performance than native solutions, such as the study by Ahti and others [2016]. However, it's somewhat difficult to compare the results, as in real applications, there are also many other factors influencing the performance of these applications besides the type of the application. Differing coding conventions and the performance of third-party libraries used in the applications can have a larger impact on the application's overall performance and overshadow the difference in the application type. In any case, if the performance of the application is the paramount requirement for the application to be developed, a native application could fulfill that requirement better than the other types of applications, with the downside of not being a cross-platform solution.

2.3 Web and hybrid applications

While the two aforementioned platforms don't support the other platform's native applications, both support the core web technologies of HTML, CSS, and JavaScript. This allows the developers to utilize the widespread web technologies in implementing applications to both platforms simultaneously. The disadvantages of pure web applications are

that they don't usually appear and work like native applications but instead need to be downloaded to the device's browser to be accessed. Because of the sandboxed browser used to run the application, they also didn't have direct access to the device's hardware, such as a camera, GPS, or motion sensors. [Raj and Tolety 2012]

To overcome these disadvantages, the web application can be turned into a hybrid application that embeds the web application inside a thin native container, which allows for combining some of the advantages of both web and native applications. Due to the native container, the hybrid applications can be packaged into a native-appearing application, which can access the underlining device hardware and data through the container's APIs. In a similar way to the web applications, hybrid applications are executed by the device's web container, so this enables the developers to use HTML5 and JavaScript to develop the applications, so no detailed knowledge of the target platform is required. [Xanthopoulos and Xinogalos 2013]

The component enabling hybrid application containers to work in the native environment is WebView. WebView is usually used by native applications when they want to load and display content from web pages on the device. However, the hybrid application container can use this same component to instead display local content written in the form of a web application. This means that instead of having an application that occasionally shows web pages through WebView, the whole application can be run in WebView, with the container supporting the web application in the background. The upside to this is that while web applications in the browser are restricted from having access to the device features themselves, WebView is a native component offered by the platform and therefore can get access to other native components and features. When native device features are needed, the web application will make a JavaScript call, which is intercepted by the WebView and passed on to the container, which will act as a bridge and maps the call from JavaScript to the native platform API and then serves the response back to the application. [Panhale 2016]

Apache Cordova and Adobe PhoneGap are examples of such hybrid application containers [Nunkesser 2018]. With the expanding capabilities of HTML5 and the browsers themselves, it has become possible to develop technology that enables developers to get many of the same benefits without requiring the hybrid application container as a middleman. The results of this development are Progressive Web Applications (PWAs), which lie in between the web and hybrid applications.

The PWAs are rendered in the device's browser like basic web-page-based web applications are. However, instead of being constrained to the browser view like a normal web page, they can be installed on the device's home screen, appearing like a native application. By using a *web manifest file*, the application can inform the device's browser that it can be installed as a PWA into the device, where a *service worker* will instruct the device on loading, caching, and otherwise running the application. [Love 2018]

Since the PWAs are installed on the device, they share some of the upsides of the hybrid applications over web applications: the appearance of a native application is often desirable, the application loads faster and can also be used offline due to caching capabilities of the service worker, and the application can run background tasks when the app is not in active use [Wargo 2020]. Therefore, the difference between the two is the underlying technology: PWAs rely on browser capabilities to access these advanced features, while hybrid applications have the underlying native container to facilitate them.

The application stores on both iOS and Android nowadays accept submissions of PWAs as well, so since a similar result can be achieved by relying on purely browser technologies, one could argue that the native container required for hybrid applications would seem redundant. However, there was no clear consensus in the literature on how these two competing technologies would fare against each other and whether the PWAs would be a possible replacement for hybrid applications. Adetunji and others [2020] recommended PWAs for mobile app developers over native and hybrid applications, but otherwise, there weren't many studies found about this topic.

2.4 Interpreted applications

The interpreted applications are similar to hybrid applications in the sense that they, too, use a separate runtime to run the application's own code, but instead of rendering the application in the web container's HTML rendering engine, they use the native SDK tools to render the UI. This means that the interpreted runtime doesn't have to be tied to the browser and web environment but can instead be anything that can run the application code and communicate with the native layer. As the rendering of the UI happens on the native portion of the app, the developers can't change the UI directly, as they could in web and hybrid applications by modifying the DOM. Instead, the interpreted app needs to communicate to the native side on runtime how the native UI widgets should be drawn to the screen to display the desired content to the user. This mapping from the tool's

abstracted UI description to native UI widgets also allows the cross-platform interpreter to change the look and feel of the app based on the platform it is running on. [Saarinen 2019]

Although interpreted applications could theoretically use different languages to manage the interpretation, JavaScript has been the common solution for the interpreted frameworks, with React Native as the dominant framework [Zohud and Zein 2021]. This means that in order to run the application, a JavaScript virtual machine is needed to interpret the application code and direct the device to display the correct native UI elements [Latif et al. 2017]. This could introduce some overhead to running the application, as the virtual machine is needed to call the native features [Latif et al. 2017]. To increase the performance of these kinds of applications, the frameworks have introduced several alleged improvements, as described in the work of Ranisavljević and others [2022], for example.

2.5 Cross-compiled applications

Cross-compiled applications differ from the previous cross-platform implementations by having the common language code written by the developer transformed into specific native byte code by a cross-compiler. Therefore, they don't need to be interpreted from a different coding language in runtime, as they have already been compiled into native applications during compilation. [Biørn-Hansen et al. 2019]

This could, in theory, give a performance edge over hybrid and interpreted applications as there is also no need for a bridging layer between the application and the native layer [Latif et al. 2017]. The native device features are also natively available, although the developer relies on the framework SDK tools to code their use in the common language code. However, after compilation, the compiled native application can naturally use these features in the same manner as any other native application could. [Biørn-Hansen et al. 2019]

There have been other cross-compiled development frameworks previously, as can be found in the works of Xanthopoulos and Xinogalos [2013], as well as in the research of Delia and others [2015]. However, many of these frameworks have since been discontinued, which is something that should most likely be considered as well when deciding which framework to use. Indeed, Nunkesser [2018] warns about vendor lock-in as well.

If the framework used is discontinued, all or most of the investments made into the development of a particular application might be in danger.

In earlier research, such as Raj and Tolety [2012], a major disadvantage to cross-compiled apps was the need to recreate the UI on every platform by the developer. This would undermine the main reason for using such a cross-platform framework since the UI code could not be shared between the different platforms. While the code for the background logic could still be shared, the additional work required by the separate UI code would understandably make cross-compiled frameworks seem worse compared to their other counterparts. However, nowadays modern cross-compilation tools can instead map the abstracted UI definitions into native UI widgets in a similar way to interpreted applications or provide their own rendering engine to render the UI in a platform-specific way when needed. [Saarinen 2019]

3 Software development practices

Software development practices have come a long way since the beginning of information system development in the 20th century. In this chapter of the thesis, a short overview of the most common practices will be done. First, we go through a brief historical overview of the development of software development practices, followed by a general description of agile practices. Afterward, the most important agile methods are described in more detail.

3.1 Traditional software development practices

The information system development can be said to have begun as early as 1940. For the first decades, it was very much the work of pioneers, after which the real push for more organized and systematic forms of software development began. The most significant early phenomenon in this was the *Software Development Life Cycle (SDLC)*. [Matkovic and Tumbas 2010]

The purpose of SDLCs was to provide a structure for the software development process by providing development tasks and methods to be used. This helps to break down the increasingly demanding and complex tasks into smaller subtasks, which in turn helps in planning and monitoring the work and facilitates cooperation and communication with the different people involved in the process. The requirements and limitations of the software development process at that point were quite different from today: the computing time was initially prohibitively expensive, so recompiling and retesting code after each minor change was simply too expensive to do regularly. This meant that manual code analysis with manpower was cheaper than testing it on the computer, so the whole process had to be constructed differently compared to today. [Kneuper 2017]

The principles from those early days have served as a foundation for numerous different development models, which can all be divided into two core categories of *sequential* and *iterative* approaches [Matkovic and Tumbas 2010]. Ruparelia [2010] also mentions the combination of these two approaches as a third category. A sequential approach has sequential stages, which always lead to the initiation of the next stage on completion. On the other hand, in an iterative model, all of the stages are revisited in the future, so the stages remain in use in the iterations throughout the development. [Ruparelia 2010]

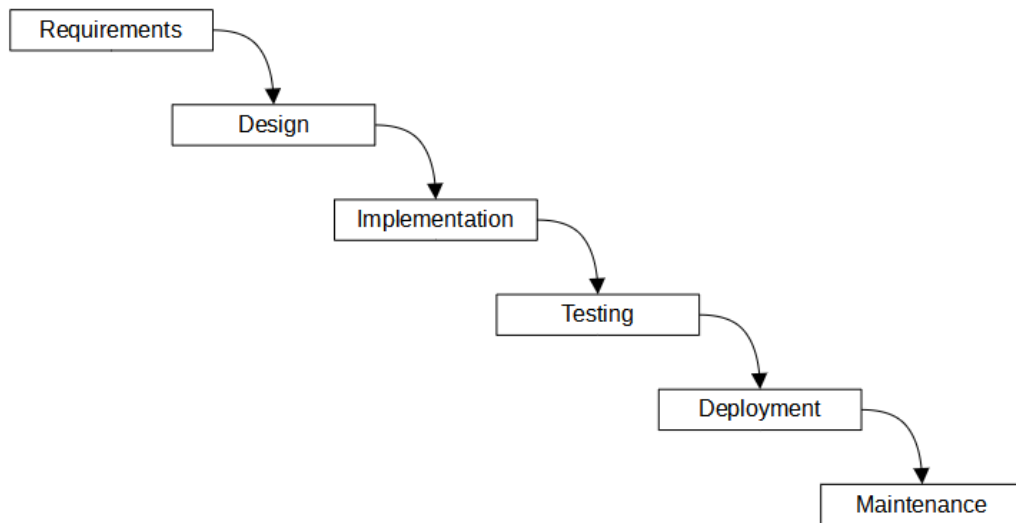


Figure 1. Waterfall model. The process flows from top to bottom like a waterfall.

Perhaps the most significant SDLC of the sequential kind is the waterfall model, which was first documented in detail in 1970 by Winston W. Royce. Its name comes from the often-used illustration, where the software development process flows in sequential steps toward a finished product, as seen in Figure 1. It was a strictly defined model characterized by standard activities described in detail for all development phases. These phases flowed from the initial requirements to the deployment and maintenance, with the design, implementation, and testing phases in between. Although the shortcomings of the waterfall model were noticed even by its author, it became the basis for many other later models and established itself as the most common model in software development history. [Leffingwell 2011]

Most of its perceived problems and shortfalls stem from its characteristic rigid and sequential structure: e.g., lack of back circuit and feedback between the stages means that the earlier stages' results are set and can't be changed easily when new complications or requirements would appear later in the development [Matkovic and Tumbas 2010]. This means that the model expects that there is a set of requirements for the project that can be reasonably determined and defined in full in the first step of the project. These requirements were then often used for the cost and schedule estimates of the project, and with the requirements, cost, and schedule all fixed in place before the project got any further, there weren't many avenues for the developers to respond to any changes or obstacles during the rest of the development. [Leffingwell 2011]

There were attempts to include feedback loops in the SDLC, but they were mostly considered forced concessions and not an approach that was deliberately planned. Until the late 1980s, iterations in the development process were mostly considered a necessary evil that was needed to correct bugs but was not actually part of the developmental life cycle. Going into the early 2000s, the emerging World Wide Web was making time-to-market grow in importance. This introduced new challenges to software development, which resulted in a need to adapt to unclear and ever-changing requirements. As a response to these changes, the use of an iterative life cycle started to gain hold, and gradually new models appeared that did explicitly support frequent iterations. Many new development methodologies were introduced, such as Feature Driven Development, Scrum, and Extreme Programming. For some time, these methods were called lightweight, as they were contrasted by the more detailed “heavyweight” processes. With the Agile Manifesto, the new term “agile” was introduced as an umbrella term and widely accepted. [Kneuper 2017]

The waterfall model or derivations of it continue to still be used today, but there has been a significant shift to agile methods (see, e.g., Leffingwell [2011], Nunkesser [2018], and Dima and Maassen [2018]).

3.2 Agile methods

The previous methods proved to be too heavy and rigid for many software development projects, with frequently exceeded deadlines and budgets in software development projects, leading to the emergence of new, more agile development methods. The fundamental idea was that the software development teams could be more efficient in change management if they were able to enhance communication between all participants in the development. This would enable them to reduce the time period between the moment a decision was made and getting feedback on the impact of said decision. The core idea is that the new corporate and technological environments require the software development process to be able to respond to changes better and faster than before. The focus was moved into a more participant-centric approach, where the participants and their individual competencies are critical to the success of the project. [Matkovic and Tumbas 2010]

Applying agile methods to very large projects can be problematic because they emphasize real-time communication between the participants [Ruparelia 2010]. On the other

hand, this makes them ideal for smaller projects, especially where participant involvement is critical. While traditional agile methods were originally designed for just one team, this has not stopped practitioners from trying to find out ways to make agile methods more scalable to larger organizations, as there are clear upsides to these methods that could potentially be beneficial to organizations of all sizes [Kalenda et al. 2018].

There are many flavors of the agile principles in different development models, which share these aforementioned core principles, but can differ significantly when it comes to the manner of realizing the actual software development process in practice [Matkovic and Tumbas 2010]. In their study about current software engineering trends, Cico and others [2021] found that of all the different software development methods, Scrum was one of the most adopted agile methods. In the same study, they also raise Kanban, Lean, and Extreme Programming as prominent agile methods used in both software development education and industry. Test-Driven Development method is also mentioned to be useful in software development education, but as the other aforementioned methods appear more frequently in the other researched literature, they were chosen for more detailed exploration in this thesis.

3.2.1 Scrum

Scrum is one of the oldest Agile frameworks, as it was created in the late 1990s, much before the publication of the Agile Manifesto. The creators of Scrum continued to participate in the creation of the Agile Manifesto as well, and it can be seen that the Scrum framework is based on many of the same ideas and values as the Agile principles that followed later.

Heath [2021] says that “*Scrum is a process framework, not a process by itself,*” which means that Scrum introduces a framework to support the development of a product, but it doesn’t opionate on how to perform the work itself: which development, technologies, or design processes to use. Scrum introduces rules, roles, and checkpoints to guide the process and encourages the adoption of agile values within the whole organization. The aim is to promote a self-organizing development team that is able to deliver working software independently while reacting to the ever-changing requirements and needs of the customer.

The three core concepts of the Scrum framework, according to Coplien and Sutherland [2019], are:

1. *Scrum Team*, with a few defined roles for the participants.
2. *Scrum Events*, short and concise meetings, that help to regularly review the progress and adapt to changes, promoting an iterative and incremental development process.
3. *Scrum Artifacts*, items that specify the work to be done and provide information about the team's progress and achievements. The artifacts are also useful for the team to develop and adapt the process and their own work.

The Scrum Team organizes and attends the Scrum Events and creates the Scrum Artifacts in addition to developing working software. There are three distinct roles in the Scrum Team:

1. *Product Owner*, who has the vision for the product and therefore can inform the other participants on the product direction, and the needs and requirements of the product and its users.
2. *Development Team*, which builds increments to the product according to the direction of the Product Owner.
3. *Scrum Master*, who is responsible for the Scrum process and leads the team to improve themselves and the process throughout the whole project.

The Product Owner and Scrum Master are roles that are usually dedicated to one person each. They have clear responsibilities and tasks defined, while the rest of the Scrum Team belongs to the Development Team, who does the actual practical development work. The Development Team can organize themselves and divide their work as they best see fit, and this should also be part of the self-improvement process that the Scrum Team will undertake during the process. There can also be other stakeholders to the process that can take part in the development process where needed but should not interfere with the Scrum process itself. [Coplien and Sutherland 2019]

The Scrum process is built upon Scrum Events, which help to pace and monitor the development process, allow the team to adapt to changing requirements, and enable continuous improvement during the process. These defined events are:

- The *Sprint*, a pre-defined interval, typically one to two weeks in length, in which the development takes place over sequential iterations.
- *Sprint Planning*, an event where relevant team members together plan the upcoming Sprint and the development work that would be undertaken during it.

- *Daily Scrum*, a daily event, where the Development Team meets to see how the whole Sprint and the work of the members are progressing, and whether there are any obstacles in the way or problems to be solved. The Sprint plan and the work can be adjusted accordingly.
- *Sprint Review*, an event after the Sprint has concluded, where the stakeholders come together to review the progress made during the Sprint and assess the current product status.
- *Sprint Retrospective*, the last but equally important event, where the team members assess their work and the Scrum process itself and analyze where there could be improvements to be made.

These events all support the core values which permeate throughout the Scrum Process: The events are time-boxed and aim to be efficient, only to use the time that is useful for the process and not to waste development time on unnecessary meetings. They focus on supporting a clear understanding of what work is currently underway and what the team will start to work on next. Regular meetings help to keep all stakeholders up to date and informed about the status of the product, and should problems arise, the team can adapt to them as soon as possible. With each iteration, both the product and the process can be refined, with the aim of arriving at the best possible result. [Coplien and Sutherland 2019]

The last of the three core concepts are the Scrum Artifacts. These are concrete items that the team collects to organize and monitor the progress of the work and refine the process:

1. The *Product Backlog* is an ordered list of items that the Product Owner expects to be delivered in the final product. While the Product Owner has the final authority over the Product Backlog items and their priority, it's usually refined together with the other members of the Scrum Team and revisited throughout the process.
2. The *Sprint Backlog* is a more detailed list of tasks that are needed to fulfill items from the Product Backlog. It's created and maintained by the Development Team and is meant to help them to plan and express how the work will be executed.
3. The *Regular Product Increment* is the result of a Sprint that the Development Team delivers at the end of each Sprint. The ideal in Scrum is that each of these increments would be in a working state, usable, and potentially releasable.

These different core concepts come together to form the basis for using Scrum in the product development process. [Coplien and Sutherland 2019]

Scrum has been widely adopted in the industry and can be considered one of the most used agile methods. In the annual survey for agile practitioners, “State of the Agile”, in 2021, 66% of the survey participants indicated that Scrum is the agile methodology that they are following most closely at the team level [Digital.ai 2022]. The popularity of Scrum is referenced in other literature as well (e.g., Leffingwell [2011], Dima and Maassen [2018], and Hooda [2023]), but it could be interesting and beneficial to conduct a more formal study too on how widespread the different agile models are in the industry.

3.2.2 Extreme Programming

As with Scrum, the idea of Extreme Programming (XP) was born already in the late 90s before the publication of the Agile Manifesto and mirrored many of the same values and goals as those two. Beck and Andres [2004] describe XP as a style of software development that focuses on the excellent application of programming techniques, clear communication, and teamwork to accomplish the best possible results. XP is based on many of the same core concepts as Scrum: short development cycles, incremental planning approach, effective communication, lightweight processes, and human interaction in the center. XP puts more emphasis on testing during the software development process and offers different tools to conduct the process when compared to Scrum.

XP opines on the actual coding practice in many ways [Warden 2003]:

- Code and design as simple as possible. Solve the customer’s current requirement, don’t try to guess future needs.
- Adopting a test-driven environment, where every time a new feature is to be introduced, first the coder writes a test for it. When the test passes, the feature is ready for refactoring. When refactoring is done, it can be added to the shared codebase. The code is supposed to be integrated very often, keeping the tasks small and not letting others wait for someone else to finish their code first.
- Pair programming is an integral part of XP. The idea is that the programmers write code in pairs, where one coder is writing the code on the keyboard focusing on the task at hand, and the other coder sits next to them, taking a larger viewpoint on how the task relates to the whole project. They do this for a singular task, after which they can find new pairs and spread the knowledge of their code to other

people. When the coders write code together, they can more easily take shared ownership of the code, follow the shared guidelines, and spread knowledge from one person to the next.

- The team should develop common coding standards and a shared vocabulary. This will help team members to understand each other's code as seamlessly as possible and keep the code more uniform.

In addition to these coding practices, there are also conventions for more general project management. XP strongly suggests taking the customer to be a part of the team. The role of the customer is to write story cards for the developers and request features for the iterations, as well as to help the team to establish acceptance tests. The *Planning Game* is introduced as a method to produce requirements and establish a schedule for the different features. The schedules in XP are based around iterations, for which the customer requests features, the team plans their execution, and they're implemented, tested, and delivered. The planning game takes place before the start of each iteration. The customer requests features by writing story cards that describe the use and behavior of the feature rather than its implementation. Then the developers take these story cards and produce estimates of the time it would require implementing these features. Then the customer gets to add these story cards and their estimates into the next iteration until the sum of the estimated time meets or exceeds the time available in the next iteration. The remaining stories go on to be fulfilled in future iterations. Finally, the development team breaks these story cards into smaller tasks, which will be worked on during the iteration. [Warden 2003]

The goal of XP is to make working releases regularly and often. At the end of each iteration, after the software has passed all acceptance tests, the software is released to the customer. The iterations are to be kept short, and this means that the customer gets to see results fast. It also allows the customer to see and test the software often, which facilitates a faster feedback loop for the developers. Possible bugs and possible changes to requirements can be detected early so the team can fix and make changes in the middle of the whole process without causing further problems. [Warden 2003]

There are also some general work practices that XP advises on. A sustainable pace of work is highlighted as one feature that promotes success in the process. If the developers are overworked, their performance will suffer, and the team will have a hard time

fulfilling their planned iteration schedule. While the customer schedules items for the iterations, it's the developer team's responsibility to inform them if some of the features will not be possible to get done during the iteration. The constant team and pair work required by XP necessitates premises that will facilitate this kind of work. An open space where the pairs can be formed naturally is ideal, while having spots where the pairs can concentrate is needed as well. In a remote working team, this can be facilitated with online collaboration software. [Warden 2003]

3.2.3 Lean

The *Lean Software Development* model borrows heavily from the theory of lean manufacturing, which was developed for car manufacturing in the 1980s by Honda and Toyota. The traditional development process product in other car companies of that time was slow, sequential, and very rigid. When decisions were made early in the development, it often wasn't possible to make any changes to them later in the process. In contrast to this, companies like Honda and Toyota preferred rapid, concurrent development, where they were able to make changes also late in the development cycle. To achieve this, the two companies decided not to make irreversible decisions in the first place, delay these decisions as long as possible, and when you have to make the decision, make them correctly by using the best possible information available. This thinking was applied to just-in-time manufacturing, where the company doesn't decide what to manufacture until they have a customer order and then fulfill that order as fast as possible. [Poppendieck and Poppendieck 2003]

The lessons learned and verified by the automobile industry with lean manufacturing raised the question of whether these could be applied to other industries as well. While software development is quite different from producing physical products in a factory, there are still a lot of practices that are applicable in this different kind of environment as well. In their book Mary and Tom Poppendieck [2003] list 7 principles of Lean development:

1. *Eliminate waste.* In the context of the automobile industry, this encompassed a lot of physically wasted material as well, but for software development, it's simply anything that doesn't add value to the product for the customer.

2. *Amplify learning.* Writing good software is not a production process; it's a development process, and so iterations can generate a lot of value for the customer. To get the best result from iteration, it's imperative to learn from the process and earlier results.
3. *Decide as late as possible.* Late decisions help to combat uncertainty in the development process. Delaying decisions can be valuable because it can enable those decisions to be based on facts instead of speculation if additional information can be gathered in the meantime. Instead of making binding decisions early, keeping design decisions open can help to adapt to changing requirements later in development.
4. *Deliver as fast as possible.* Rapid development provides many opportunities: without speed, you cannot delay decisions, and you can't gather reliable feedback. Short development cycles enable continuous feedback from the customer, and more information will be gathered. As said, they also enable postponing decisions until possible changes are more likely to be known.
5. *Empower the team.* Making most of the decisions late and working fast means that not everything can be decided by a central authority. The previous principles can only be achieved with a dedicated and motivated team. Equipped with the necessary expertise, the team can be guided by a leader to make better technical decisions and process decisions than what could be decided by any one person.
6. *Build integrity in.* Integrity in software means that it has a coherent architecture and is therefore maintainable, adaptable, and extensible. For the users, it needs to score high in usability and fitness for purpose, to fulfill what needs and requirements the user has for it. A good foundation for building this integrity is using relevant expertise, effective communication, wise leadership, and healthy discipline.
7. *See the whole.* Often experts in any area tend to focus on their own specialty rather than focusing on the overall system performance. If every contributor focuses on their own small piece of the overall picture, a suboptimal overall performance is a likely result. Complex systems consist of numerous smaller parts, and if we concentrate on optimizing only these smaller pieces, the whole will most likely suffer. Taking the whole also into account can result in better overall performance.

The Lean Development method offers much broader instructions on how to run a development project than Scrum or XP; there are no set roles or events and no specific practices to follow. Instead, it offers these listed principles that can help to find the best practices for any given specific situation and team. The idea is that principles are universal, and every team can decide how to best apply these principles to their work. [Poppendieck and Poppendieck 2003]

3.2.4 Kanban

In a systematic literature review of Kanban practices, Al-Baik and Miller [2015] found that Kanban was put into use in a few different ways in the studies included: Kanban was either seen as 1) an element of Lean methodology, 2) an agile methodology itself, 3) a part of a hybrid methodology of Kanban and Scrum, or 4) a stand-alone tool without a direct link to either agile or lean methodologies. Therefore, it is possible to describe a process to be using Kanban while actually having a different perspective from another practitioner stating the same. Therefore, it's useful to speak of Kanban also as a separate concept from the Lean methodology, even though both share many of the same values and are often applied together. The beauty of Kanban is that it can be used in itself as a project management tool, but it also pairs well with other agile ideas and methodologies.

The Kanban system has the same background as the more general Lean methodology in the Japanese automobile industry. They introduced a visual board of cards (lit. *kan* "visual" and *ban* "cards") to the production line, where these visual cards represent tasks in different stages of the production and when each of them is completed and ready for the next stage. One of the most important things to make Kanban work is to limit the number of cards allowed on the board at the same time. This is accompanied by a pull system, where the later stages pull cards forward only when there is space, ensuring that the amount of work in progress in the system remains manageable. [Anderson 2015]

According to the work of Anderson [2015], there are six core practices that help to make a project using Kanban successful:

1. *Make work visible.* In the center of Kanban is the Kanban board, which helps to visualize the workflow. This helps to make the whole work process clear to all team members and makes it easy to track how the work is progressing and if any problems are occurring along the way.

2. *Limit work in progress.* Even from the viewpoint of traditional production management, unfinished products are tied-up capital, and only finished products bring value to the customers. By keeping the amount of work in progress minimal, the individual tasks get finished faster, enabling continuous improvement. If all tasks are unfinished, there isn't much to learn from any of them. Increasing unfinished work also brings difficulties in managing the whole project, while getting tasks finished helps to move the project forward.
3. *Manage flow.* The focus of the Kanban board is on the workflow: how the tasks are progressing in the flow. Everything that seems to slow or block this flow requires special attention from the team. The principle is to solve these problems first before starting to work on something else. Fortunately, the board is an easy visual tool to spot these sorts of problems and try to solve them before they start to affect the whole process.
4. *Make progress policies explicit.* When the rules for marking a stage of a task ready are mutually agreed upon, it's easy to uphold the level of quality throughout the process. Everybody should have a shared understanding of what needs to be done in the different stages, and therefore if this is not done, the peers can give feedback based on the agreed-upon policies.
5. *Implement feedback mechanisms.* There should be an implicit focus on achieving continuous improvement as a team. Therefore, it is imperative to have a feedback mechanism to help the team members to improve, as in order to be able to learn something, we often need feedback to identify how we can do things better.
6. *Improve collaboratively.* As mentioned in the previous core practice, Kanban holds continuous improvement as a highly important principle. As the team members can improve their own work and help others do the same, the team can also improve together. All of these can be achieved by working feedback mechanisms but also by hosting retrospectives to analyze previous work.

To put it very simply, Kanban helps teams to map, manage, and visualize their workflow. The goal is to keep the process very light and leave most of the working time to actually produce value for the end users. However, Kanban seeks to make the most out of this apparent simplicity. By setting limits on how many cards can be pulled into a stage on the board, the work stays manageable, and everybody can understand where the project is going. By setting explicit finishing conditions, everybody knows what is required for

the card to be finished in one stage and be ready to be pulled for the next stage. [Anderson 2015]

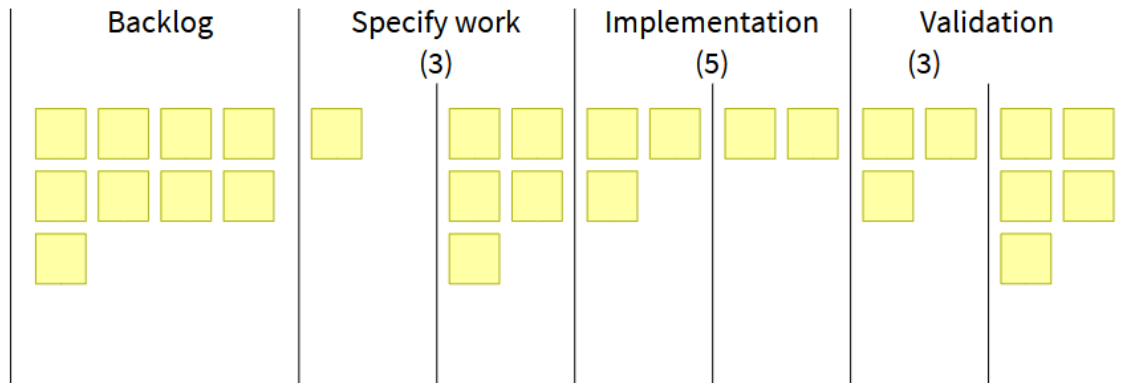


Figure 2. Example of a Kanban board.

Figure 2 gives an example of a Kanban board, where the yellow blocks represent tasks moving through the stages. The numbers under the stage names represent how many cards can advance into each stage, and this limit should not be exceeded. Other stages except the backlog have been divided into two columns, with the left being “work-in-progress” and the right being “ready”. The workflow has been divided into four stages:

1. The backlog: stores the story cards which have not yet been taken into the process.
2. Specifying the work: in this stage, the items from the backlog get broken into smaller tasks that should be more equal in size than the larger backlog items. Therefore, the team can estimate how long a single card should take up time, and if this time is exceeded later in the process, it can be identified and remedied. As the cards from the backlog get broken down into smaller tasks, the number of cards in this column can exceed the work-in-progress limit set for it. This is not a problem, but it means that no new items can be pulled from the backlog if there are too many cards waiting to be pulled to the next stage. This prevents the next phase from being overwhelmed with the number of tasks waiting for them. It also signals to the team that no more specification work should be done right now, but instead, the next stage needs more people working on it if possible.
3. The implementation: where the actual implementation work takes place. The previous stage should have specified the work in such a manner that this stage knows what they need to achieve in order to implement this feature, in addition to fulfilling the agreed-upon conditions on when an item can be marked as ready.

4. Validation: Last stage, where the implemented cards get validated before entering the “work done” column. Note that the work-in-progress limit only applies to the left column here, as the right side contains all cards that have been finished.

This is only one example of a possible Kanban board; especially when the team gets larger, the board should be expanded both horizontally and vertically. Vertical expansion can be implemented by raising the “work-in-progress” limits, while horizontal expansion can be achieved by adding more stages to the workflow. In any case, the workflow of the organization should map into the stages of the board, but making the stages too small can introduce unwanted friction to the process. [Anderson 2015]

There is only one type of event required by Kanban: the daily standup. Any member of the team can run the daily standup, but if there is a project manager or similar in the team, it’s probably often the most natural choice. The only question required to be on the meeting agenda is whether any team members are blocked in their tasks or otherwise need assistance. Often team members also like to learn what other members are or have been doing and to celebrate their progress in the project. [Anderson 2015]

4 Developing mobile applications in a startup

What is a startup? There is no exact agreed-upon definition, but usually, a company can be considered to be a startup when they exhibit most of these characteristics [Sutton 2000]:

- *Youth and immaturity*: the most basic characteristic of a startup company. They are relatively new and inexperienced and thus have very little accumulated experience or history.
- *Limited resources*: very often, startups work with extremely limited resources, which need to be used on outward-looking activities, such as building the product, promoting it, and building connections for the company. Accomplishing these tasks will be vital for the company to survive in the long term.
- *Multiple influences*: in the early stages, a company might be particularly sensitive to various differing (and possibly contradictory) requirements and influences. While the company is still looking for direction, it might continue to adjust and readjust its course based on these influences.
- *Dynamic technologies and markets*: startups are often the first ones to take up new technologies, markets, or other possibilities. While they might appear fragile when compared to bigger, more established companies, their characteristics also make them ideal for making the first move on new opportunities.

One obstacle that almost all startups need to face is uncertainty, which appears in many different areas: market, product, competitiveness, finances, and people. In fact, the whole process of creating a new business is recognized as difficult, complex, and risky. All this uncertainty makes investors, potential employees, and customers hesitant to providing resources to a startup, which means that startups need to find ways to succeed despite the lack of economic, human, and physical resources. [Bortolini et al. 2021]

Many of the studies about startups conclude that a great majority of startups fail, and there is a multitude of possible causes for their eventual failure (see, e.g., Blank [2013], Giardino et al. [2016], Cantamessa et al. [2018], or Bortolini et al. [2021]). Many of the aforementioned studies come to the conclusion that the Lean Startup model by Blank [2013] could help startups to address some of the challenges that often lead to the failure of the startup. The Lean Startup model incorporates many of the lean principles

discussed earlier and accommodates them in the context of startups. Shepherd and Gruber [2021] list five building blocks as the foundation of the Lean Startup framework:

1. *Finding and prioritizing market opportunities*: finding the right need and place in the market is imperative if the startup wants to be able to compete, create value, and achieve viability.
2. *Designing business models*: when a startup finds the right market opportunity, it's equally important to find the right way to conduct business in that domain.
3. *Validated learning*: when founding a startup, the founders have some initial hypotheses about the business. These hypotheses need to be tested and validated, and this learning needs to continue throughout the startup journey.
4. *Building Minimum Viable Products* (“MVPs”): MVP is a version of a product or feature that enables a full turn of the build-measure-learn loop but requires the minimal amount of effort possible. This enables agile development, where the startup can get into iterating the product as fast as possible.
5. *Learning when to preserve and when to pivot*: The iterative nature of this framework enables the startup to conduct trial-and-error learning on their product. However, sometimes the startup might recognize that the incremental changes do not generate enough improvements to justify continuing and might therefore need to “pivot” to make a larger course correction in order to test something different instead.

The principles of the Lean Startup thus serve as a good foundation when examining the different prospects for a startup and how they could advance or hinder the startup on the journey to success.

Despite the harsh constraints and risks listed above, the startups are also in a unique position to undertake new and innovative projects that might be too risky or unconventional for more established companies. This allows startups to potentially find new market areas that were previously either ignored or underutilized or create products that better correspond to the needs of the customers in that area. This means that the impact of startups is significant in creating new opportunities and businesses for the economy. In their systematic review of the impact of entrepreneurship on economic, social, and environmental welfare, Neumann [2021] found that new firm formations have a generally positive impact on regional economic and societal welfare, while the effect on environmental welfare depends on other conditions. Therefore, supporting the right kind of

startups can offer desirable effects for the economy and society, from the governmental or societal point of view as well.

4.1 Feasibility of different technical approaches for startups

Most of the time, startups try to search for solutions that provide fast results with the least resources, the time-to-market being one of the primary goals [Klotins et al. 2019]. When a startup is at the beginning of its journey, it's important to get an MVP fast in order to move forward [Lee and Geum 2021]. Before this has been achieved, most of the time, a startup has very limited resources, so cost-effectiveness and development time optimization are of the utmost importance. Once the MVP has been developed, the startup can move forward with testing the product-market fit and more easily search for additional funding.

When considering these needs, the native applications can often be too resource-intensive to develop. Usually, startups want to target both big mobile platforms (Android and iOS), and in order to do this with native applications, they would need to develop separate codebases for the different platforms. Web applications can be one solution if the focus audience is on traditional desktop environments, and mobile applications are just additional services for the customers. However, this is a very lightweight solution for mobile platforms, so many of the upsides of native applications will not be available when developing a web application. In this spectrum, the hybrid, interpreted, and cross-compiled applications fall in the middle: they provide the ease of a single main code base for all platforms while still functioning almost like a native application in most of the other aspects.

Often the final decision is a compromise between the requirements of the application and the resources available to the startup. If the startup founders or team members have experience in one of the possible technologies, if it otherwise fits the needs of the startup, it's often the easiest choice. Developing on an already familiar platform has several natural upsides, mainly faster development, which is often one of the main priorities of startups. Careful consideration should be given to the matter, as almost everything else in the startup depends on the development team succeeding in creating the desired application. [Drongelen 2017]

While the startup might build the first MVP with one technology and keep developing that, at some stage, it might come to a point where the startup might decide to

abandon the first application and build a new application from scratch with another technology. This might be necessary, for example, if the technology picked is deemed unsuitable for the requirements of the startup or if the technical debt accumulated during the development has reached a level where continuing development would necessitate very extensive rework to the codebase. Or maybe during the testing of the MVP, the startup would find that they need to make such large changes to the basis of their application that it's going to be easiest to start again almost from scratch.

While the decision to do so would be difficult with the already scarce resources, knowing when to preserve and when to pivot is one of the core principles of the Lean Startup framework. When this kind of decision is based on the hypothesis testing and testing of the earlier version of the application, it can be done with minimal effects on the startup and the team morale. On the other hand, failure to either test the application sufficiently before piloting or continuing to pour resources into a failing business model will have a strong negative effect on the continuation of the startup. The Lean Startup model emphasizes that startup founders need the courage to make the decision to pivot when it is found to be necessary, as failure to do so will most likely cause the startup to fail in the process. [Shepherd and Gruber 2021]

4.2 Software development practices in a startup

Startups are often very creative and flexible in nature and therefore reluctant to adopt bureaucratic, laborious, or restrictive processes, which might hinder their biggest assets. As mentioned before, with the limited resources that they have, startups often like to dedicate their limited resources to product development instead of establishing processes. Product-oriented practices leave them with the ability to quickly change direction according to the needs of the target market. Research on software development in startups often emphasizes the importance of time-to-market, so the development practices are also tailored to its needs. A peculiar notion compared to the more established companies is that the requirements are often reported to have been “invented by the company itself”, “rarely documented”, and “can be validated only after the product is released”. These can lead to the product not meeting the customer's needs, which can lead, in the worst case, to the failure of the startup. [Paternoster et al. 2014]

With these constraints and requirements, it's not very surprising that agile methods have been considered the most viable process for software startups. Many of the agile

methods analyzed in this work as well have been experimented with in startup companies. Other agile themes are often explored as well, but often the startups' processes do not strictly follow any specific methodology but instead opportunistically select practices that fit their needs. [Klotins et al. 2019]

In many cases, the more formal software development methods are forgotten for the sake of moving fast. In their study, Kemell and others [2020] noted that in some cases, startups had taken a method such as Scrum and then immediately omitted some practices from it to create their own version of it. Even more frequently, startups seem to forgo even the use of these tailored methods, pointing to a highly unsystematic approach to software development. While the absence of even basic processes might enable startups to concentrate more on the product itself, startups could still benefit from tailoring at least some simple development method practices to their needs. One such example was found in the study by Taipale [2010], where a startup reported gaining benefits from applying some of the simpler XP practices to their workflow.

Giardino and others [2016] proposed the *Greenfield Startup Model (GSM)* to showcase the underlying phenomenon of software development in early-stage startups. In this model, they don't aim to dictate the best practices or give guidelines to the startups, but instead, it's meant to showcase the conditions and practices with which the startups need to work with. GSM once again notes that the speed of development is paramount for startups. Without a working product, all startups would fail eventually. To be flexible, reactive, and fast, startups often try to leverage their team's capabilities by keeping the software development process simple and the workflows informal. The fact that the teams are most often self-organized, and the developers have significant responsibilities within the company facilitate this kind of behavior. This type of fever-paced development restricts potential planning activities, so in order to deal with this kind of unpredictability, startups often prefer to make decisions as fast as possible and adapt them later if needed. Even though agile principles should fit into this kind of change almost perfectly, startups often perceive formal development practices as a waste of time and prefer to ignore them in favor of spending all of their time in getting the product to the market as fast as possible.

Other notable findings in the GSM [Giardino et al. 2016] include that startups often prefer to utilize known technologies and standards that are supported by strong communities. This enables the startups to reduce the time used in formal architectural design and creating conventions for themselves when they can find these from these ready-to-use

packages. Startups also often accumulate technical debt during their fast-paced initial development by ignoring the aspects related to documentation, structures, and processes.

Giardino and others [2016] summarize their findings about startups into seven main categories:

1. *Severe lack of resources*: startups generally lack resources in multiple key areas: time, human resources, and access to expertise. Of these, the lack of time is usually the direst, as startups strive to get their product to the market as fast as possible.
2. *Team is the catalyst of development*: with the lack of human resources, the developers have big responsibilities in startups, causing them to be active in every aspect of the development process. Coders in the founding team are typically full-stack developers and often need to fulfill additional roles, such as marketing and sales. Having a very small and co-located team enables high coordination within the team, replacing most of the documentation with informal discussions.
3. *Evolutionary approach*: startups often build an initial prototype and iteratively refine it further with the goal of validating the product-market fit. The companies focus on building a small set of basic functionalities with the least possible effort to validate them with user feedback and continue iterating the prototype further.
4. *Product quality has low priority*: with the limited human resources and time shortage, the startup needs to prioritize their work, and often product quality gets prioritized lower. Of the quality aspects of the development, the effort is directed towards the user experience: “If the product works, but it is not usable, it doesn’t work.”
5. *Speed up development*: Simple and informal workflows allow startups to be flexible and reactive, always adapting to a fast-changing environment. While agile methods would embrace change, startups often perceive formal development practices as a waste of time and ignore them in favor of releasing the product to the market faster.
6. *Accumulated technical debt*: The high speed of development often accrues technical debt by radically ignoring many of the more traditional aspects of development related to documentation, structures, and processes. While accumulating technical debt can speed up development initially and be crucial for a startup’s

survival in the early stages, it can start to slow down the development after a certain point.

7. *Initial growth hinders performance:* The lack of attention given in the first stages of startups allows them to release the first prototypes quickly. However, if the startup survives the early stages, the initial product starts to become more complex over time. At some point, the accumulated technical debt forces the development team to address the accumulated technical debt instead of focusing on developing new features. Thus, the initial growth achieved can later hinder performance in terms of new functionalities delivered to the users.

From these findings, we can note that startups need to balance difficult extremes in their software development processes. On the other hand, there is a great need to get the product out to the market as fast as possible, but often the means to achieve that can produce other complications for the startups later in their path. While the benefits of practices that accrue technical debt can be useful in the short term, over time, technical debt has a negative impact on morale, productivity, and product quality. Finding the balance between these and solutions that facilitate the best outcomes is vital for maximizing the possibility that the startup will survive the turbulent early stages. While the consequences of technical debt might not be evident in the initial stages of the startup, where finding the product-market fit is the most important priority, eventually, the startup needs to pay back the accumulated technical debt. [Giardino et al. 2016]

Using this data, Giardino and others [2016] constructed a theory about the development strategies of early-stage software startups: These startups operate at a fever-paced development speed, focusing on a limited number of core functionalities. They generally employ partial and rapid evolutionary development approaches, striving to find product-market fit as fast as possible while under uncertain conditions and with a severe lack of resources. However, often they speed up the process by accumulating technical debt, causing an accumulating decrease in performance. This is done with the intent of helping the startup to get to the point where they would have the resources and capability to start paying back the technical debt.

This theory and their findings led to a list of implications that could be considered when trying to map the best development practices for startups:

1. *Light-Weight Methodology*: Early on in the development, the startup understandably concentrates almost all of its energy on developing the product. In a situation like this, startups tend not to apply any specific or standard development methodologies. Despite this, taking advantage of some of the simpler development method practices could be beneficial for a startup. Especially later, once the initial chaos has been managed, there should be planning for adopting Agile and Lean development practices.
2. *Empowering the Team Members*: Team empowerment is described as a critical factor for pursuing the MVP, and it can positively impact the team's performance as well. Giardino and others [2016] suggest the following empowerment programs:
 - a. autonomy to make decisions and pick activities they are interested in,
 - b. responsibility for the organization's results or success,
 - c. keeping track of their own performance,
 - d. team members able to have an influence on the decisions made,
 - e. promoting creativity and a culture where negative results are not punished but attempts are rewarded.
3. *Focus on a Minimal Set of Functionalities*: To be able to deliver with their extremely limited resources, startups need to prioritize the important functionalities and filter out others. While formal requirements engineering wouldn't be conducted, integrating simple techniques such as Persona and Scenario can help startups to gather data about the requirements more effectively.
4. *Paying Back the Technical Debt*: Even though technical debt can be useful in the short term, over time, the accumulated technical debt can start to slow the project down. Tracking technical debt can help to manage it, as well as the usage of modern coding platforms and well-known frameworks. It's also useful to consider when the technical debt is the most useful and when avoiding it will be better overall. At some point, it might be necessary to start paying the technical debt back, hopefully at a point where the startup can support it.

In their systematic mapping study on agile development practices in software startups, Abrantes and Furtado [2021] found that many startups do indeed follow agile practices in their development. They cited DevOps, Tests, Scrum, and XP as the most

frequently found agile practices among startups. Only Scrum and XP were discussed in the previous chapter of this thesis, as the other two are not considered software development methods. However, they would likely still be useful agile practices for a generic software startup: DevOps is often used in conjunction with agile methodologies to improve the speed and quality of the development by utilizing integration and automation in the development process and delivery of the software. Tests, on the other hand, refer to the assortment of different testing methods and tools that the developers can use to improve code quality and the speed of development by helping them identify and fix errors more quickly.

Abrantes and Furtado [2021] bring forward many of the same challenges identified in GSM [Giardino et al. 2016] as well: startups often report having insufficient resources available and different organizational challenges, such as the pressure of time-to-market pushing startups to move forward very fast in their development. The immaturity of the companies was also reported as one of the problem-causing areas for startups. Overall, they found that startups prefer speed-related practices to quality-related ones, which fits the pattern of startups ranking speed of development as one of the highest in their priorities.

5 Action research on developing LessonApp

To explore these concepts further, a study into mobile application development in a startup is conducted. The motivation behind the study is that since startups present unique challenges for software development activities, it could be beneficial to research the topic also in a real-world setting, to see how well the theoretical background translates into practice in a particular instance. My own work in a startup brought me an opportunity to study this phenomenon in the field.

First, the research methodology used is discussed in more detail in Chapter 5.1. Then the startup and its current state of application development are introduced in Chapters 5.2 and 5.3. Lastly, the action research process in action is described in Chapter 5.4.

5.1 Research methodology

Using the taxonomy of research methods by Järvinen [2021], the object under study here is a startup, which is “a part of reality (a system)”, and the goal of the research is to “try to achieve utility”, by evaluating and improving the practices in the said startup. Therefore, the chosen research method should be *action research*. Action research seems a very suitable method for this study, as there are problems to be solved in the startup in question, and making changes to their existing conventions is possible, so we can implement and see how these changes affect the mobile application development in this startup. For action research, the participation of the researcher in the actual work done in the startup is not necessarily a problem if the limitations and possible biases are recognized and accounted for. In his dissertation, Järvinen [2021] presents the Canonical Action Research (CAR) by Davison and others [2008 and 2012] with additional remarks and improvements as a way to ensure rigor when conducting research with the action research methodology. These guidelines will be followed in this action research case as applicable.

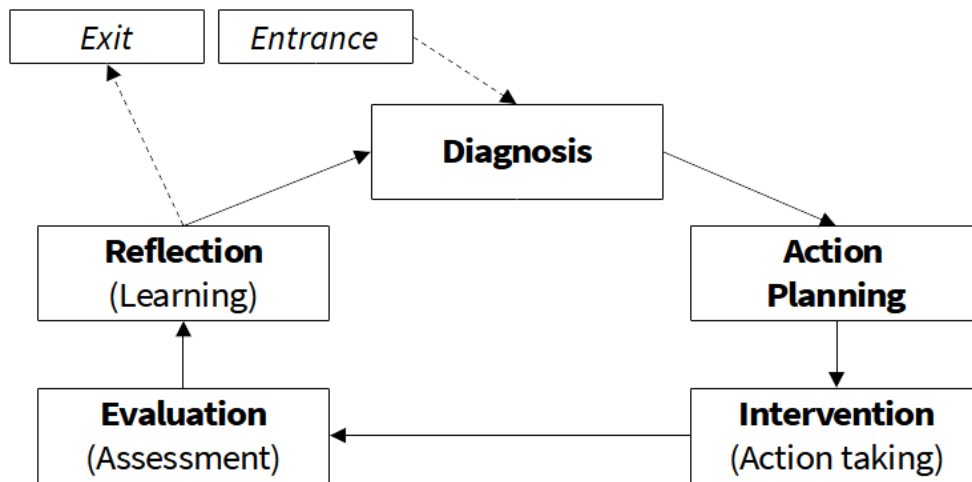


Figure 3. The cyclical Process Model for CAR [Davison et al. 2008].

Action research involves addressing organizational problems through intervention while at the same time contributing to the scientific community. Already from the early stages of action research, the process has been cyclical in nature: first, identify the problems and plan actions to address them, then take action according to the plan. Afterward, evaluate the effects of the actions taken and reflect on the results. Finally, use these findings to repeat the cycle, starting with analyzing the problems remaining or created during the action and planning new actions accordingly. This cycle is illustrated in Figure 3. [Davison et al. 2008]

Davison and others [2008] proposed the following five principles for CAR:

1. *The Principle of the Researcher-Client Agreement*: ensuring that the client targeted by the research understands how CAR works and how it will affect the organization.
2. *The Principle of the Cyclical Process Model (CPM)*: the action research should be cyclical and follow the steps in the CPM.
3. *The Principle of Theory*: as applicable, the planned actions should be grounded in the theoretical framework found in the literature.
4. *The Principle of Change through Action*: the essence of action research is to implement actions to change the current situation for the better. These changes have to be planned and well documented, both beforehand and their effects afterward.
5. *The Principle of Learning through Reflection*: while the actions taken should produce improvements for the client, the learning that can be enabled by this process

must not be neglected, but instead, reflection on the actions taken should be used to gain knowledge to satisfy the research side of the action research.

The literature review about mobile application technologies and software development methodologies works as the theoretical basis for the intervention in the action research case. The current state of both was evaluated in the startup in question, and actions were planned accordingly. Two cycles of the action research cycle were planned for this thesis, where in between, the first changes were evaluated, and new actions were planned for the second cycle. In the end, an evaluation of the effectiveness of the actions of the startup in question was done.

5.2 About LessonApp

LessonApp is an application for teachers and trainers to plan their lessons using a large catalog of different teaching methods and with an innovative lesson-planning tool. LessonApp is available as an app for Android and iOS, as well as a web application for web browsers. In LessonApp, the teachers can plan their lessons using building blocks that represent different parts of the lesson and choose appropriate teaching methods for each block from the library of methods. These lessons can be saved for personal use but also shared with colleagues within their school or with all the other users of LessonApp. LessonApp also contains pedagogical material to learn more about the Finnish way of teaching.

The company was founded in the spring of 2018, so at the start of this research, LessonApp had operated for a little over four years. The founders consist of 5 people, the CEO, two pedagogical experts, one web developer, and one full-stack developer. Over the years, there have been several other coders working for different periods of time, but currently, there are two people working on the code: one of the founders and a new team member. The company is managed jointly by the founders, and all major decisions are made together in the founders' meetings.

The research project collects observations from both the development team and the other members of the startup team. The development team which analyzes the technical implementation consists of the researcher and another coder. The pair also participates in meetings with two other members of the startup team who specialize in pedagogical development. Together, this team monitors the development and assesses the development methods used.

5.3 The current state of development

The startup in question already has an application published for public use that they had created prior to this project. The current application was implemented using pure HTML and JavaScript, supplemented with jQuery, and then packaged into a native application with the use of Apache Cordova. While this provided an easy way to create the first prototype relatively fast and the way the team wanted it to look like, when the project continued, it soon became apparent that this approach would make expanding the scope of the application relatively cumbersome. Therefore, the startup decided that it could be beneficial to use some of the other available frameworks to build the application again from the beginning while considering the newly emerged requirements from the earlier development.

The development process until this point has not been systematic but instead very product-oriented, making rapid changes based on the changing needs of the product and customer feedback. The current product shows some signs of technical debt and age, such as quickly fixed parts instead of comprehensive solutions to the actual problems, non-consistent use of programming conventions, and a sprawling codebase. This has been mostly a result of attempts to adapt to changing requirements in the middle of the development, which have not been taken into account earlier in the development cycle. There have also been different contributors to the code during the development, and the lack of clear conventions has led to different conventions being used in different parts of the application. There are also numerous instances of code duplication, where the same piece of code has been re-used separately in various parts of the program, which greatly increases the work required to make any changes to that code.

No formal development methods have been used during the development. For most of the development, there has been only a single developer or two, so it was thought that there would not be much to gain from more formal project management structures. As the resources for the development have been very limited, it was thought that all the available time should be allocated to writing more code for the application, not to bureaucratic processes. This approach resulted in producing the initial application relatively fast, but the development process has been slowing down since.

The current state of LessonApp corresponds quite accurately to the descriptions of startups found in the literature as well. As characterized by Giardino and others [2016] in

GSM, LessonApp too has been operating in a chaotic, rapidly evolving, and uncertain environment, which has been affecting the software development of their product as well. The startup has accrued a lot of technical debt during the development that is currently slowing the development down. Some of the decisions made before and during the development of the application have led to negative consequences on the long-term performance, maintainability, and extensibility of the product. Therefore, the company stands in a position where the team has agreed that there are strong reasons to make a pivot from the current practices.

5.4 Conducting the action research: first cycle

Before beginning the first cycle, the Principle of the Researcher-Client-Agreement of CAR was considered.

5.4.1 Diagnosis

The team had voiced concerns that the old technology stack could be potentially unsuitable for continued use in the future. During the independent diagnosis of the situation, the researcher found that there were several reasons for this. The development of new features with the current technology stack was relatively slow. There were a lot of existing bugs and short-term solutions that would have required great contributions of work to be corrected. There was also a considerable amount of technical debt acquired from the rapid pace of development and poorly followed conventions. It was also found that the team didn't follow any defined software development methodology or framework while organizing the development work but instead relied on the discretion of the development team and individual coders to deliver requested features in a reasonable timeframe.

Implementing an intervention related to these problems would potentially yield findings about the suitability of the different solutions to the startup and provide other insights about the usage of technologies and software development methods in startups.

5.4.2 Plan

Based on these findings, a radical change to the development conventions was proposed. Adopting a new cross-platform technology and developing a new second version of the application with this new technology could help the team address the shortcomings of the current technology. In this process, the team would move to a more formalized form of development by adopting one of the agile methodologies to their work and adapting it in

a way that suits their needs. The team would define more exact coding conventions to use while developing the new version of the application, and the team would then commit to following these practices. While the development would still continue in a swift manner, the team would pay special attention not to resort to short-term solutions or code duplication but would try to follow better coding conventions agreed upon within the team.

While the team was implementing these changes, the researcher continued to monitor the development process by participating in both the development itself, as well as in the meetings held by the team. For the research part of the process, it was important to document how the team carried out the planned changes and whether they were realized in action as planned. The effect of the changes to the development process was also of particular interest, whether the changes resulted in improvements for the team or were deemed not suitable, and what other findings could be gained from these actions.

5.4.3 Act

Together with the team, a decision to adopt Flutter as their new technological base for the application was made. Flutter is an open-source framework for building cross-compiled applications for mobile phones and web browsers. Flutter is built around widgets that can be reused around the application. It is, therefore, a simple way to reduce code duplication, something that had caused problems for the team earlier. Flutter is based on Dart, which is a strongly typed object-oriented language. Dart itself provides a style guide that is recommended to be followed, especially if one wants to share their code with the open-source community. The team decided that they would adopt these styling rules while writing the code and would define their own conventions for the cases that the style guide doesn't cover.

For the software project management side of the equation, there were many different options, but many of them could be hard to apply to a development team of just two members. While the other members of the company were valuable in providing requirements, feedback, and practical testing, they could not contribute to writing the code for the application. Presented with the different options, the team felt that many of them would be excellent choices with a slightly bigger team, but they would be difficult to apply to the team's current situation with only two people available to work on the code.

Based on these discussions with the team, it was decided that the team would apply Kanban principles to the development team's work and modify the Kanban board to better suit their workflow with only two people.

As was planned, the researcher took part in the daily work of the startup while the team was carrying out the planned actions. As a member of the team, the researcher took part in both the development of the application, as well as in the meetings of the startup team. The efficacy of the planned actions was assessed by both observing the implementation and effects of the carried-out actions. The team members were able to express their observations and thoughts about the changes during the team meetings, and open discussion about the changes was encouraged. In addition to this, the researcher conducted short interviews with the team members about the changes made and how the team members felt that those had affected their own work and the startup in general. The findings and observations were written down and verified with the team members to ensure that the researcher had correctly understood their thoughts and written them down accurately.

5.4.4 Evaluate

The team acted on the planned actions for two months, after which the situation was evaluated. Based on the observations of his own work, as well as the team meetings and interviews, the researcher assessed the team's situation and the effects of the intervention to be the following. Flutter was deemed to be working well for the team and for building a new version of the application. By following the style guide and agreed-upon coding conventions, the code produced by the team was much more consistent than what was found in the previous work of the startup. Of course, the situation at this point is very different, as the team is rewriting the application from the beginning with new technology instead of adding new features on top of old code, as was the case with the old version of the application. However, the team still felt that it would be much easier for them to keep producing better code than before, as the new coding conventions adopted had made it much clearer for everyone how to keep the code base coherent.

The team adopted some of the Kanban principles to their work and used cards on a virtual board to define individual work tasks and monitor the progress of the work. However, despite the intention to follow these principles closely, the team took some shortcuts in their work and said in their own evaluation that they didn't get to fully utilize what Kanban had to offer. In their opinion, the Kanban board could most likely provide much

more for the team if the team size was just slightly larger. With just two coders working on writing the code, it often made the most sense for one of the coders to take care of a work task from start to finish, so different stages of the task were not utilized very much, as would be the governing idea in a Kanban board. Still, the team found Kanban principles, and especially the task cards, to be helpful tools for tracking their work but felt that the model itself didn't necessarily fit their needs perfectly.

The intervention was evaluated to be moderately successful from the team's point of view and successful from the researcher's point of view. While the team's development process would most likely still require some additional changes after the first cycle, already at this point, the observations produced by the process should be useful for the research and wider audience as well.

5.4.5 Reflect

This Kanban-inspired workflow allowed the team to work very freely: the developers had only a list of work tasks that needed completion and were free to take on any task that seemed appropriate to them at the time. If there were difficulties in a certain work task, the developer was free to either leave the task to rest for some time while working on something else or move the task to the other developer in case they could find a way to resolve the issue. The developers praised this freedom as motivating and easy to work with. However, in many practical cases, there might be much more narrow objectives for the work and not so many available tasks to choose from. Also, when considering slightly larger teams, this extent of freedom might make the task allocation difficult to manage. In this case, more meticulous adherence to the Kanban constraints would probably be necessary and helpful.

Flutter as a technological framework provided better tools for building a cross-platform application for the team than the previously used technology stack. The team found the hot reload function especially useful for the development work, where smaller changes made to the program could be loaded into the application while it was running on any of the platforms. This enabled the team to see the effect of changes fast and experiment with different solutions and was especially helpful when building the UI for the application.

The research process during the first cycle worked relatively well for gathering information and observations about the research topic. For the second cycle, closer and

more frequent interaction with the team members outside the development team could help to bring up some insights or observations that were not yet found during the first cycle. For the first cycle, the focus of the research had been mostly on the development team, but in order to understand the whole startup team better, it would be beneficial to involve the other members of the startup team more in the process as well.

5.5 Conducting the action research: second cycle

After the completion of the first cycle, the researcher assessed the situation, and it was decided by the researcher, together with the team, that they would continue into a second cycle of CAR.

5.5.1 Diagnosis

While the developers expressed their satisfaction with their new technology Flutter, some areas were found where the new technology could be utilized better. Even though the developers had found the platform to be helpful in maintaining a consistent coding style, the two developers used different *integrated development environments* (IDEs) for writing the code, and this resulted in some discrepancies between the developers.

Although the ideas presented by Kanban were thought to be useful and sensible for this kind of development work, the team did not adopt the practices fully, taking advantage of the parts that they thought were sensible for their situation. This resulted in some benefits for the planning of the work, but the team felt that it would be prudent to experiment with another agile development method to compare with this earlier experience.

One additional problem identified at this stage was that the team members who were not part of the actual development were somewhat disconnected from the development work. While the development team was working closely together, there was less communication with the other members of the team, who were not taking part in the daily development work. The team felt that there was a need for more inclusion of the other team members in a way that doesn't take too much of their time from other work but keeps them updated on the state of development.

From the research point of view, continuing to monitor the work of the startup could bring some additional findings related to the further changes made to the development

process of the startup. Adopting another software development method would allow us to compare the two and how they were received by the startup in question.

5.5.2 Plan

To address the pitfalls identified on the current technological solution, it was decided that the team should take steps to better enforce the shared coding practices and the style guide while writing new code. During the development, more emphasis should be placed on making the code easily readable and expandable. The discrepancies between the different IDEs should be remedied so that there are no major differences in code written between the two.

For the development practices, the different agile methods were examined in order to find a method that would fit the team's needs. From the first round of CAR, it was found that the cards from Kanban were useful for the team in mapping the different work tasks and their current status in the workflow. While some of the team members valued the freedom that the team had adopted to the workflow, it was decided that it might be useful to have some more guiding structures in the management of the work tasks. The team also agreed with the researcher that increasing communication between the development team and the other team members would be necessary for the future. To address these needs, it was decided that the team should adopt Scrum in their work.

The researcher would continue to work with the team in a similar fashion to the first cycle. The efficacy of the implemented solutions would be monitored and evaluated again, comparing the current situation to both the situation before the action research, as well as after the first cycle of it.

5.5.3 Act

The development team decided to install the same formatting plugins to both of the IDEs, which helped to harmonize the coding style between the coders. Some of the styling choices were also discussed, and the team agreed to adopt these in their work with the help of the newly adopted formatting plugins. The team continued to create widgets for different kinds of needs in the code, with the intention that these widgets could be used application-wide in different places without the need to create them anew every time.

As was planned, the team started to use Scrum to manage the development work. One of the developers was chosen to be the Scrum Master, tasked to train the other members on how to be a part of a Scrum team. One of the team members outside the development team was chosen to be the Product Owner in order to bring the viewpoints of the other team members better into the development work. The Sprint Review meetings also provided an easy way to keep the other members of the team up to date with the development status of the application. The length of the Sprints was decided to be rather short, usually just one week, with the intention of making it possible to make rapid changes to the newly developed features before having to commit to a solution.

No major changes were made to the research process in the second cycle. The researcher continued to monitor and gather observations from both his and his team's work. This was done by actively writing down found observations, interviewing the team members, and cross-checking with them that all essential findings were correctly documented.

5.5.4 Evaluate

The changes made to the coding practices were deemed useful, and the team decided to continue using them in the future as well. The formatting tools assisted the coders in keeping the code base more uniform and readable. The development team noted that some of the new code was harder for the other coder to understand, so some of the code was restructured or rewritten so that it would be clearer and more understandable for future use as well. The team also identified some older code pieces that were against the coding conventions laid out by the Flutter guidelines. These breaches of the guidelines could result in decreased performance while running the application, so the team determined that some more code upkeep and refactoring would be needed in the future.

Scrum was assessed to be a useful software development method for the team's needs. The practices included in Scrum were well received by the team, and the team adopted them more closely than in the previous stage. The Sprints were held with the length of one to two weeks, after which the team arranged a Sprint Review meeting. This more organized way of development and regular team meetings promoted better communication within the team, especially between the development team and the other team members. These sprint review meetings were kept short so as to not use too much of everyone's time on the meetings. These meetings were received very favorably, and in

addition to keeping everybody informed of the status of the product, they let the development team discuss matters related to different features of the product with the other members of the team in an organized manner. As the development work progressed, the team realized that there was a need for a more direct line of communication as well, so better means of direct messaging between the team members were introduced as well.

One of the objectives of adopting Scrum was to better involve the team members outside the development team in the development of the application. This objective was deemed to have been very successful, as the team members felt that by using Scrum and having regular meetings in the form of Sprint Reviews, the non-coder team members were effectively engaged in the application development. The interaction within the startup team became more regular and effortless, and the development process was transformed into a shared project where different problems were easier to solve together. Any decisions to be made regarding the functionality and the features of the application got more diverse viewpoints when the larger team got to consider them together in the meetings. Overall, the regular meetings and letting the other members of the team get more involved in the development process inspired them to advance the development of the startup in other areas as well. Finally, the education experts of the team valued the continuous learning aspect of Scrum as well: “Cooperation and genuine teamwork were an important part of the Scrum for us and enabled the team to learn and improve together.”

Despite mostly adopting the Scrum framework, as was described earlier in the thesis, there were still some changes made to the traditional model by the team. One clear deviation from the Scrum model was that the team agreed that the Sprint plans made before the start of each sprint were not quite as set in stone as the Scrum would necessitate. In most of the literature, it was stated quite strongly that the Sprint plans should be locked in, and no additional work should be introduced in the middle of the Sprint. While there are clear reasons why this is the recommended way, the team determined that there were some specific reasons why this was not practical for the team’s situation. Although usually, this practice would promote successful Sprints, where the planned work is completed before the end of the Sprint, in our startup’s case, the team thought that in case the planned work was completed before the end of the sprint, it wouldn’t be reasonable to wait until the end of the Sprint to start working on something else. With a bigger team, someone running out of work would mean that they could help other team members with some unfinished work or otherwise try to promote a successful Sprint, but with this small

team, getting more work done was deemed to be more important than adhering strictly to the Sprint plan.

Of the other parts of the Scrum framework, the Sprint planning session and the Sprint Backlog were valued the most. Regular planning of the Sprints helped the development team to assess the amount of work required to complete different features and will serve them in future planning as well. The Product Backlog was updated regularly at the start of this cycle but was somewhat neglected later in development when the team somewhat supplanted the board with the discussions in the Sprint Review meetings. Regardless of this, the team acknowledged that the Product Backlog would still be a useful tool in the future, so the team should find better procedures to keep it updated henceforward. The development team started the Daily Scrum as a regular thing, but it was also somewhat waning towards the end of the cycle. With just two developers working closely together, it's often quite clear what the situation with the other one's work is at any given time. In the future, it needs to be assessed whether taking some time every day to conduct a formal Daily Scrum brings enough benefits that the team should continue using it.

Although the first cycle was found to be rather successful from the research point of view, the second cycle provided even more findings to the research than the first one. This was mainly due to the increased interactivity with and within the team. With the more regular meetings, it became easier to observe the team working together and gather their observations more often during the process as well. While most of the knowledge to be gained from the technical side was probably gathered in the first cycle, continuing to observe the work with the new technology and interviewing the team members brought up more findings that were not discovered in the first cycle.

5.5.5 Reflect

The support of the formatting tools in keeping the coding style consistent was found to be very useful for the development work. While there were some styling choices that the team had some reservations about, the ease of having the IDE enforce the style guidelines exceeded these small drawbacks. During the cycle, the team didn't encounter any problems caused by the automatic formatting, but it's something to consider when using automated coding tools. One important observation for the team from this cycle was that while the focus of this cycle was on the style of coding and harmonizing it between the

coders, there is also a diverse range of other issues that need to be recognized when considering improving coding practices. There were several other issues identified that could affect the quality of the code and increase technical debt if they were not addressed in the future. While the team had strived for consistent naming schemes and structures, there would still be a need to better define the way the code should be written so that it would be easier in the future to understand the old codebase. The team had employed limited comments on the code and very limited additional documentation, which were also identified as points to improve in the future. This would be especially important if the development team wants to grow beyond the initial two developers.

One of the developers reflected that the Scrum model was easier to adhere to, as it was a good fit with the manner and pace of the development that the team had earlier organically developed for themselves. The development team still had a rather free reign on what they wanted to work on next, while Scrum at the same time introduced structure and directions to the work. This helped to make the development more organized while not pushing the developers into a development style that they would not be comfortable with. Therefore, it's evident that a critical component of achieving change in an organization is getting the team members' "buy-in" to the changes. If the changes don't make sense to the team members, they are less likely to abide by the desired changes, and they might easily either fall back to their old ways or selectively modify the changes to fit better to their ideal.

Overall, the team was satisfied with the changes to both the coding practices and the software development management. While they made their own adjustments to the Scrum model, the core practices were still more closely followed than in the previous cycle. These new practices helped the team to organize their work in a more efficient and sustainable manner by their own evaluation. Therefore, the team plans to continue using these practices in the future as well.

Continuing a second cycle of action research was useful for the research as well. In theory, while sometimes one cycle could be enough to implement the desired changes, continuing to monitor the target organization for multiple cycles can bring additional insights that would not be possible to acquire in a singular cycle of research. And, of course, continuing to refine the process by implementing new changes helps to find the best pos-

sible solution to be reported in the research findings too. Allowing the researcher to compare and contrast the changes to the ones in the first cycle should also provide additional findings that couldn't be produced without the continuation of the cyclical process.

5.6 End of the second cycle

With the second cycle of CAR ending, the researcher decided to conclude the research project. While the startup will continue to develop its way of working in the future as well, it was decided that the formal research cycles would end at this point.

6 Results

There are a lot of potential aspects to be studied in the development work of startups. This thesis focused on two of those aspects: the technology used for making mobile applications and the software development methodology used to facilitate it. In the first two Chapters, 6.1 and 6.2, the results from the literature review are discussed from the point of view of the first two research questions. After this, Chapter 6.3 concentrates on the results of the action research and strives to answer the third research question.

6.1 Modern mobile application development frameworks for startups

While there are numerous different technological solutions for making mobile applications, they could be separated into four categories: native, web and hybrid, interpreted, and cross-compiled applications [Raj and Tolety 2012]. For most startups, the need to get the product to the market as fast as possible with minimal resources means that the native applications are not practical, so they often end up choosing technology from one of the three latter categories.

Native applications often offer the best performance, with the downside of requiring separate codebases for the platforms [Ferreira et al. 2018]. For some very specific use cases, where the performance of the application is critical, this might be necessary, but for most cases in startups, the overhead of maintaining different versions of the application slows the development too much to be feasible [Xanthopoulos and Xinogalos 2013].

The first cross-platform solutions were web applications, as the core web technologies are supported by all platforms. The downside of pure web applications was that they didn't look and feel like native applications, as they needed to be downloaded to the

device's browser to be accessed. Because of the sandboxed browser used to run the application, they also didn't have direct access to the devices' hardware, such as cameras, GPS, or motion sensors. [Raj and Tolety 2012]

To overcome these disadvantages, hybrid application frameworks were developed. They allow web applications to be embedded inside a native container, which allows for combining some of the advantages of both web and native applications. The native container allows the web application to appear as a native application and access the devices' hardware through it while still enabling the developers to use the web technologies to develop a single codebase. [Xanthopoulos and Xinogalos 2013]

With the expanding capabilities of HTML5 and web browsers, it has been possible to develop technology that enables applications to get many of the same benefits as hybrid applications without requiring a hybrid container as a middleman. This development has resulted in Progressive Web Applications, which lie in between the web and hybrid applications. The PWAs are rendered in the device's browser like basic web-page-based web applications are. However, instead of being constrained to the browser view like a normal web page, they can be installed on the device's home screen, appearing like a native application. [Love 2018]

In addition to the different web browser-based solutions, interpreted and cross-compiled applications are the last two categories of available cross-platform solutions. Interpreted applications use a separate runtime to run the application's own code, which communicates with the native SDK tools to render the UI. While this produces native-appearing applications with a single codebase, the separate runtime can introduce some overhead to running an application [Latif et al. 2017].

Finally, cross-compiled applications differ from the other solutions by compiling the developers' common language code into native byte codes for the different platforms with a cross-compiler. This produces native applications for the platforms from a single codebase, which could, in theory, result in better performance than the hybrid or interpreted solutions [Latif et al. 2017].

There are different positive and negative sides to all of these solutions, but often a startup will make the final decision as a compromise between the requirements of the

application and the available expertise. Careful consideration should be given to the matter, as the future of the startup is largely dependent on the team being able to produce the desired application. [Drongelen 2017]

6.2 The best practices for the software development process in startups

While previously, the most used software development methods were sequential like the Waterfall method, startups have been quick to adopt iterative agile methods to facilitate their needs for quick development with ever-changing requirements. There are many different agile methods for startups to choose from, but Scrum, Extreme Programming, Lean, and Kanban have been some of the most used development methods in startups [Cico et al. 2021]. However, often startups' processes do not strictly follow any specific methodology but instead, opportunistically select practices that fit their needs [Klotins et al. 2019].

Scrum is a framework that introduces rules, roles, and checkpoints to guide the development process, promoting a self-organizing development team that is able to produce working software and react to ever-changing requirements. The Scrum workflow is organized around Sprints, defined intervals of time, during which the development takes place over sequential iterations. Together with the team, the Sprints are planned beforehand, the progress is monitored, and the results are presented and evaluated afterward, with the goal of improving the process itself and the self-improvement of the participants all through the development. The Product Backlog and the Sprint Backlog are used to plan and monitor the development activities, with the target of producing a Regular Product Increment after each Sprint. [Coplien and Sutherland 2019]

Extreme Programming (XP) is a style of programming that focuses on rigorous programming techniques, clear and effective communication, and teamwork [Beck and Andres 2004]. Some of the more unique properties of XP include a test-driven approach to the development, constant refactoring of the written code, and pair programming. Similar to Scrum, XP works with sequential iterations, where the team works in close collaboration with the customer and produces working releases regularly and often. [Warden 2003]

The Lean Software Development model offers much broader instructions on how to run a development project than Scrum or XP. Instead of very specific roles and practices, it lists principles that are thought to be universal, and every team can then decide

how to best apply them. The list of 7 principles of Lean development describes these principles in detail, but the core idea is to minimize waste in the process, develop the product rapidly, and empower the team in the process. [Poppendieck and Poppendieck 2003]

The last agile method discussed was Kanban. While Kanban can be utilized as a stand-alone tool, it's also often paired as a hybrid method with Scrum or seen just as a part of the Lean methodology [Al-Baik and Miller 2015]. Kanban revolves around a visual board of cards that is used to plan, showcase, monitor, and track the workflow of the company. This makes the work items and their status visible and also controls the workflow and the amount of work currently in progress. [Anderson 2015]

The Greenfield Startup Model [Giardino et al. 2016] showcases how for startups, it's essential to get the product out to the market as fast as possible. However, often when trying to achieve this, they might accrue technical debt that will, over time, have a negative impact on the product quality and further development. Finding the balance between the extreme requirements and limitations of the startup and the solutions that facilitate the best outcomes is vital for maximizing the possibility that a startup will survive the difficult beginning. Accruing technical debt can be acceptable and even in some sense desirable if it helps the startup to survive the tumultuous beginning, but monitoring, managing, and rationing technical debt is needed in order to not let the development grind to a halt because of it.

Giardino and others [2016] emphasize the severe lack of resources in startups and the need to accommodate this in everything that the startup does. Prioritizing the work, speeding up the development, and enabling the team to achieve their full potential all contribute to the efficient use of the limited resources. There are several ways to empower the team members: increased autonomy, increased influence in the whole process, promoting creativity, and fostering a culture where negative results are not punished but attempts are celebrated. All of this helps to motivate the team members and can positively impact the team's performance as well. Keeping the development methodology light is almost mandatory, but startups could still benefit from adopting at least some simpler practices to their work. Using evolutionary approaches to development helps to effectively exploit the limited resources to build only what is necessary and concentrate on the most important features.

6.3 The best practices in practice in a startup

The above-mentioned insights from the literature review were applied to practice in a startup called LessonApp. Many of the characteristics of a generic startup held true with LessonApp: the development had been rapid, and the application had accumulated substantial amounts of technical debt. The developers were using some of the agile principles in the development work, but no formal software development methods were used. To address the problems identified in the work of the startup, several actions were planned and executed during two rounds of action research.

The development team of LessonApp started using a cross-compiled application framework Flutter to develop their application. They experimented with adopting Kanban for their work but found Scrum to be a better fit for the current needs of the team. After ending the second research cycle, the team had adopted these two changes deeply into their everyday work and planned to continue using them in the future as well.

One interesting feature of this startup case was that the first MVP was developed quite extensively, to the point where it was no longer just an MVP but a product that the company was selling. This meant that the decisions that the developers had made with the MVP in mind were still affecting the product. Like Giardino and others [2016] described startup activities in their work, the developers had sacrificed a lot of technical quality in order to get the product out as fast as possible. While there were some attempts at structuring the code, no actual frameworks or tools were utilized, which meant that the codebase soon became very hard to maintain and extend. Realizing this, the startup agreed to start from scratch and plan the development better for their new round of development. While it seems quite impossible for a regular startup to start anew and build the application again from scratch, in this case, also in hindsight, the startup considered this to have been an excellent decision. The pains caused by the old codebase could now be left behind, and the future of development seems much more optimistic. And maintaining ease of development is crucial for the startup in the future as well because the product is still far from complete, and they will need to keep adapting the product extensively in the future as well. With too much technical debt, this would be impossible, but with the new codebase, the technical debt has been reduced substantially.

This kind of radical change to the startup will not be possible in many cases, but it would be vital for a startup to be able to recognize if the situation had developed to the

point where this kind of radical intervention would be needed. Only time will tell how the new solutions and the startup examined here will fare, but at the time of writing, the startup team felt that the decision to make the change was ultimately a necessary one, and the changes resulting from it were beneficial for the startup, especially for the future.

There were several notable observations brought up from the action research cycles. Using a coding language with its own opinionated style guide and a formatting tool to enforce the chosen coding style helped the different coders to keep a consistent coding style and will help to keep the code maintainable and expandable in the future as well. And with the basic conventions laid down by the coding language and formatting tools themselves, the time input required by the coders can be lessened. While spending nearly any time on planning coding conventions and documenting them might seem frivolous for a startup starved of resources, forgoing them completely will lead to complications, as seen from the earlier work of the startup. Conversely, the time used to establish these kinds of conventions can start to save time already early on in the development, as was found in practice here. This follows the same pattern as described in the GSM [Giardino et al. 2016], where startups are said to prefer using standards and well-known technologies, which enable them to reduce the time needed to develop their own conventions and standards.

Finding the right software development method that suits the team well is essential for making the development work productive, coordinated, and effective. While again, the time-saving measure that would be easy to implement would be to just leave the developers to work freely on what is deemed the most important at any given time, as the startup had done previously, this will lead to problems down the road and will be unmaintainable if the startup wants to grow beyond one or two developers. Creating a culture where the time spent in planning and coordinating the development work is not considered wasted will help in maintaining a consistent planning routine. This also means that especially in a startup context, where the resources are extremely limited, it's of utmost importance that when time is spent on these kinds of activities, the use of time is carefully planned and executed so as to not waste any of the participants' time. This will help keep the meetings and other planning activities relevant and useful instead of a resource sink that will not help to promote the success of the startup.

While there are many well-known agile software development methods available, in our startup's case, using Scrum as the basis for the development was deemed the most

suitable. And as was found in the literature as well (e.g., Klotins and others [2019]), the startup tailored the development method to what suits them the best. Although there are usually good reasons for established conventions in these development methods, startups do differ from more established companies in many regards, so this kind of behavior is not entirely unexpected. When the startup matures, it's probably a good time to take into consideration what kind of reasons the startup had to make the deviations to the established conventions and whether they still remain beneficial to the company and the development team. One key finding from this research was also that experimenting with different development methods can be very useful for a startup. While the Kanban-inspired development method was considered an improvement to the original way of development by the startup, exploring Scrum in the second cycle of the research allowed the startup to establish a development method that they felt was much more suitable for their context. If there is a willingness to experiment and adapt new ways of working, it's possible to find new ways to manage the development work, which can considerably improve the old conventions.

Based on the results of the action research, it appears to be potentially beneficial for a member of the startup team to familiarize themselves with the different agile development models available and try to find a solution that would work the best for their team in their current situation and the foreseeable future. Taking one of the agile methods and applying it into the current context of the startup could help the startup to organize their work better and therefore lead to better overall results for the startup as well.

Finally, it was found that participating in a study like this was mutually beneficial to both participants, the author, and the startup in question. As Davison and others [2004] expressed in their work, both CAR and action research in general aim to help address organizational problems while at the same time contributing to the scientific community. Conducting a rigorous and critical inquiry into the organization can help to uncover and address problems that the organization itself could be unable to identify or address. In this case, the startup team itself had recognized that its current situation was undesirable. However, only with the research project and the intervention were there serious attempts to change and improve the situation. The changes made during the research cycles and the whole intervention itself were viewed as successful by both parties.

7 Conclusions

This thesis concentrated on two major themes from a startup point of view: the different mobile application technologies available and different software development methods that can be utilized in developing them. In the first half of the thesis, a literature review was conducted on these topics. Afterward, the theoretical background was used in an action research study with a startup. During the study, the startup developed a new version of their application with a new technology Flutter and explored different possible software development methods.

Although there are numerous different possible topics to conduct research about startups, these two themes were chosen as they were identified as being two large problems related to software development for the startup participating in the research. While these might not be sources of problems for all startups, all software startups need to address these topics in one way or the other. Therefore, the findings from this particular research and the startup's case could be of use to them. While it's impossible to claim that these findings would be universally applicable, they provide a unique perspective from a startup that could be useful to other startups facing similar questions. Additionally, if more research is conducted on a similar topic, the results from this research could serve as both a source of information and a point of comparison for future researchers.

The results of the action research showed that a cross-platform solution like Flutter worked well for this case. The cross-platform functionality enabled a single well-functioning codebase, and the code formatting tools, together with the style guidelines included in the platform, were practical for keeping the codebase consistent and maintainable. Lastly, the developer tools and the widget-based coding style helped to prevent code duplication by allowing the developers to reuse the widgets in many instances. These were previously problematic for the startup with the old codebase, so these improvements, combined with the renewed fast pace of development, were certainly useful for the startup. The software development method in the startup was first changed from a free-form development to Kanban and later to Scrum, which was found to be the most suitable solution for this team. For each of the development methods, the startup team made changes to the model to better suit their situation, which appears to be common among startups in the literature as well.

As a startup, LessonApp expressed many of the characteristics of startups found in the literature: the pace of the development had been very fast, the company had accrued technical debt in order to get to market faster, and the company was going forward with very limited resources. Many of the articles in the literature described a typical software development process in a startup as “exploring agile themes, but not strictly following any specific methodology,” and this was the case with LessonApp as well. However, during our action research, we found that for this team, adopting Scrum as the software development method of choice was an improvement over the previous practices. Nonetheless, characteristically to the startups, the LessonApp team also modified some of the Scrum practices to better suit their team’s needs.

In this case, researching the different agile methods and experimenting with two of them led the team to find a basis for the software development process that they felt fitting their team and their workflow much better than the previous more unorganized development model. Based on these findings, we would encourage other startups to consider doing this as well. Finding a suitable software development method can help to organize the development much better and can lead to improved outcomes for the startup. And as described in this thesis as well, the startup in question can freely modify the practices to better fit their team, or try out another development method, if the first one is not deemed a success.

Another notable finding from the action research was that the team felt that the decision to build a new version of their app from scratch was ultimately the right decision to make. While Shepherd and Gruber [2021] suggest that startups need to know when to pivot, oftentimes, the pivot is not quite as radical as in this case. Leaving behind all the work done on the first version of the app could seem wasteful, but at the same time, this move that the LessonApp team made allowed them to reset the technical debt that they had accumulated so far. Starting from scratch allowed the team to take the lessons learned from the first version and put them to use in building the second version without the burdens of the earlier development weighing them down. This kind of radical pivot might not be possible in all cases, but where it is and is deemed necessary, we highly recommend considering it.

7.1 Limitations

While the active participation of the researcher in the research process is an intrinsic part of action research, it's still something that should be acknowledged when examining the possible limitations of the research. Especially in this case, when the researcher himself was fully participating in the daily work of the startup, this is definitely a matter of concern. Working in the double role of the researcher and both the software developer and the co-founder of the startup could have an impact on the process and perceived results. Analyzing your own work can be influenced by being too critical or too lenient on the perceived faults or strengths of the work performed. It's also quite possible that there are some aspects of your own work that you've grown too familiar with to analyze objectively. And while this could be to some extent addressed by gathering feedback from the other participants of the project, the personal relationships between the co-workers can affect their perception and the feedback that they give on the changes made during the action research.

While personal bias cannot be fully ruled out, the researcher has strived to describe the whole process as accurately as possible and work around any personal biases that could interfere with the individual steps of the action research cycle. Essentially all of the steps of the action research cycle contain subjective analysis, but this would be almost impossible to avoid, even if it were desirable to do so. On the other hand, this kind of intimate work inside the target organization can yield findings that would be impossible to obtain through other research methods.

One possible avenue of trying to mitigate the personal bias in the research could have been to try to measure some more objective indicators of the effect that the technology and the development methods were having on the performance of the startup. Measuring the amount of code written, commits made, bugs encountered, or time used for different tasks could be possible ways to try to evaluate the changes in performance, but finding the right way to measure these would probably be a good starting point for another thesis. Therefore, as this is such a complex concept to measure, the researcher opted to rely on the discussions with the startup team to consider what kind of effect the team perceived to happen with the changes made. This included the other coder of the development team, as well as the two other members of the startup who took part in the Scrum meetings. Therefore, the observations about the technology were always based on the pair discussions with the other coder, reflecting on the advantages, disadvantages, and issues

arising from the technology. Observations about the development methods, on the other hand, were gathered from the team of four participating in the development process.

With the sample size of one startup, the results found here of course cannot be generalized for the wider startup sphere. Instead, the purpose of this research was to provide the reader with insights and perspectives from a practical experience of one startup. Action research has also often been criticized for its lack of methodological rigor, but to address this, the principles of canonical action research (as laid out by Davison and others [2004]) were applied as applicable.

7.2 Future work

There is still an abundance of possible topics to be researched in startups in general, as well as in their software development practices. For future research, it would be most worthwhile to conduct additional comparable studies into other startups, to get more in-the-field knowledge about the state of startups and their software development. As Carroll and others [2022] found in their study, the field of software startups is a rapidly growing area where there have been considerable research efforts recently, but the after-effects of the Covid-19 pandemic are still affecting the research efforts tremendously.

While this thesis concentrated on the technology behind mobile applications in a startup and the software development methods used to develop them, there are many interesting possible avenues of research in closely related areas. The front-end technology is of course a core part of the user experience and, therefore, tremendously important for the startup, but there are many other technical questions that the startup needs to address as well. Whether the startup needs to create a back-end server for the application, a cloud environment, or a serverless solution, often the data of the users has to be handled in one way or the other. Add into this the user authentication, sending notifications to the users, or how to create a publishing pipeline, and we can conclude that startups need to consider exorbitant amounts of different questions and solutions when it comes to building a complete application. Studying and comparing different kinds of solutions from the startup point of view could be immensely helpful for upcoming startups and could help to understand this side of startups better. Of course, one problem is that technology in this area is developing very fast, and it's hard for research to keep up with the current inventions, but this only serves to highlight how difficult it is to find up-to-date research on these topics.

During this research, it also became evident that there are a lot of other difficulties in addition to the technical problems that a startup faces constantly in their day-to-day work while trying to navigate their way to a successful company. The whole concept of a startup is very multifaceted, so some more interdisciplinary approaches could help to investigate these kinds of topics further.

References

- Paulo Cesar Abrantes and Ana Paula Furtado. 2021. Agile Development Practices Applied to Software Startups: A Systematic Mapping Review. In: *2021 16th Iberian Conference on Information Systems and Technologies*, 1–6.
- Oluwatofunmi Adetunji, Chigozirim Ajaegbu, and Olawale Jacob Omotosho. 2020. Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development. *American Scientific Research Journal for Engineering, Technology, and Sciences* 68(1), 85-99.
- Ville Ahti, Sami Hyrynsalmi, and Olli Nevalainen. 2016. An Evaluation Framework for Cross-Platform Mobile App Development Tools: A case analysis of Adobe PhoneGap framework. In: *Proceedings of the 17th International Conference on Computer Systems and Technologies (CompSysTech '16)*, 41–48.
- David J. Anderson. 2015. Kanban Principles and Core Practices. In: Siegfried Kaltenecker and Klaus Leopold, *Kanban Change Leadership*. John Wiley & Sons, 8–24.
- Osama Al-Baik and James Miller. 2015. The Kanban Approach, Between Agility and Leanness: a Systematic Review. *Empirical Software Engineering: An International Journal* 20(6), 1861–1897.
- Andreas Biørn-Hansen, Tor-Morten Grønli, and Gheorghita Ghinea. 2019. A Survey and Taxonomy of Core Concepts and Research Challenges in Cross-Platform Mobile Development. *ACM Computing Surveys* 51(5), 1–34.
- Steve Blank. 2013. Why the Lean Start-up Changes Everything. *Harvard Business Review* 91(5), 64–.
- Rafael Fazzi Bortolini, Marcelo Nogueira Cortimiglia, Angela de Moura Ferreira Danilevicz, and Antonio Ghezzi. 2021. Lean Startup: A Comprehensive Historical Review. *Management Decision* 59(8), 1765–1783.
- Kent Beck and Andres Cynthia. 2004. *Extreme Programming Explained: Embrace Change*. 2nd ed. Addison-Wesley, 2004.
- Marco Cantamessa, Valentina Gatteschi, Guido Perboli, and Mariangela Rosano. 2018. Startups' Roads to Failure. *Sustainability* 10(7), 2346–.

Noel Carroll, Anh Nguyen-Duc, Xiaofeng Wang, and Viktoria Stray. 2022. The Evolution of Software Startup Research: A Survey of Literature. *Software Business* 463, 304–319.

Orges Cico, Letizia Jaccheri, Anh Nguyen-Duc, and He Zhang. 2021. Exploring the Intersection Between Software Industry and Software Engineering Education - A Systematic Mapping of Software Engineering Trends. *The Journal of Systems and Software* 172, 110736–.

James O. Coplien and Jeff Sutherland. 2019. *A Scrum Book*. Pragmatic Bookshelf, 2019.

Anuradha Dande, Veli-Pekka Eloranta, H. Hadaytullah, Antti-Jussi Kovalainen, Timo Lehtonen, Marko Leppänen, Taru Salmimaa, et al. 2014. *Software Startup Patterns - An Empirical Study*. Report 4. Tampere University of Technology, Dept. of Pervasive Computing.

Robert Davison, Maris G. Martinsons, and Ned Kock. 2004. Principles of canonical action research. *Information Systems Journal* 14(1), 65–86.

Robert Davison, Maris G. Martinsons, and Carol X. Ou. 2012. The Roles of Theory in Canonical Action Research. *MIS Quarterly* 36(3), 763–786.

Lisandro Delia, Nicolas Galdamez, Pablo Thomas, Leonardo Corbalan, and Patricia Pésado. 2015. Multi-platform mobile application development analysis. In: *IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, 181–186.

Digital.ai. 2022. 15th Annual State of Agile Report. Referenced 10.9.2022. <https://digital.ai/resource-center/analyst-reports/state-of-agile-report>

Alina Dima and Maria A. Maassen. 2018. From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. *Journal of International Studies* 11(2), 315–325.

Mike van Drongelen. 2017. *Lean Mobile App Development: Apply Lean Startup Methodologies to Develop Successful iOS and Android Apps*. Packt Publishing. 2017.

Cristiane M. S. Ferreira, Maria J. P. Peixoto, Paulo A. S. Duarte, Andrei B. B. Torres, Messias L. Silva Júnior, Lincoln S. Rocha, and Windson Viana. 2018. An Evaluation of

Cross-Platform Frameworks for Multimedia Mobile Applications Development. *IEEE Latin America Transactions* 16(4), 1206–1212.

Carmine Giardino, Nicolo Paternoster, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson. 2016. Software Development in Startup Companies: The Greenfield Startup Model. *IEEE Transactions on Software Engineering* 42(6), 585–604.

Fred Heath. 2021. *The Professional Scrum Master (PSM I) Guide*. Packt Publishing. 2021.

Susheela Hooda. 2023. *Agile Software Development: Trends, Challenges and Applications*. John Wiley & Sons. 2023.

Tomislav Jakopec and Tena Vilček. 2017. Comparative analysis of tools for development of native and hybrid mobile applications. In: *40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1516–1521.

Pertti Järvinen. 2021. *Improving guidelines and developing a taxonomy of methodologies for research in information systems*. Ph. D. Dissertation, Faculty of Information Technology, University of Jyväskylä.

Aleksander Kaczmarczyk and Wojciech Zabierowski. 2021. The Comparison of Native and Hybrid Mobile Applications for Android System. In: *28th International Conference on Mixed Design of Integrated Circuits and System*, 290–293.

Martin Kalenda, Petr Hyna, and Bruno Rossi. 2018. Scaling Agile in Large Organizations: Practices, Challenges, and Success Factors. *Journal of Software: Evolution and Process* 30(10), p. e1954.

Kai-Kristian Kemell, Ville Ravaska, Anh Nguyen-Duc, and Pekka Abrahamsson. 2020. Software Startup Practices – Software Development in Startups Through the Lens of the Essence Theory of Software Engineering. *Lecture Notes in Computer Science (including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12562, 402–418.

Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. 2019. Software Engineering in Start-up Companies: An Analysis of 88 Experience Reports. *Empirical Software Engineering: an International Journal* 24, 68–102.

Ralf Kneuper. 2017. Sixty Years of Software Development Life Cycle Models. In: *IEEE Annals of the History of Computing* 39(3), 41–54.

Mounaim Latif, Younes Lakhri, El Habib Nfaoui, and Najia Es-Sbai. 2017. Review of mobile cross platform and research orientations. In: *International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, 1–4.

Sungjoo Lee and Youngjung Geum. 2021. How to Determine a Minimum Viable Product in App-Based Lean Start-Ups: Kano-Based Approach. *Total Quality Management & Business Excellence* 32(15-16), 1751–1767.

Dean Leffingwell. 2011. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley, 2011.

Chris Love. 2018. *Progressive Web Application Development by Example: Develop Fast, Reliable and Engaging User Experiences for the Web*. Packt Publishing, 2018.

Predrag Matkovic and Pere Tumbas. 2010. A Comparative Overview of the Evolution of Software Development Models. *Journal of Industrial Engineering and Management* 1, 163–172.

Thomas Neumann. 2021. The Impact of Entrepreneurship on Economic, Social and Environmental Welfare and Its Determinants: a Systematic Review. *Management Review Quarterly* 71(3), 553–584.

Robin Nunkesser. 2018. Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development. In: *IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 214–218.

Mahesh Panhale. 2016. *Beginning Hybrid Mobile Application Development*. Berkeley, CA: Apress, 2016.

Nicolò Paternoster, Carmine Giardino, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson. 2014. Software Development in Startup Companies: A Systematic Mapping Study. *Information and Software Technology* 56(10), 1200–1218.

Mary Poppendieck and Michael A. Cusumano. 2012. Lean Software Development: A Tutorial. *IEEE Software* 29(5), 26–32.

Mary Poppendieck and Tom Poppendieck. 2003. *Lean Software Development: An Agile Toolkit*. Pearson Education, 2003.

C.P Rahul Raj and Seshu Babu Tolety. 2012. A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: *2012 Annual IEEE India Conference (INDICON)*, 625–629.

Tamara Ranisavljević, Darjan Karabašević, Miodrag Brzaković, Gabrijela Popović, and Dragiša Stanujkić. 2022. React Native: a brief introduction to modern cross-platform mobile application development. *Quaestus* 21, 120–136.

Nayan B. Ruparelia. 2010. Software Development Lifecycle Models. *Software Engineering Notes* 35(3), 8–13.

Jarkko Saarinen. 2019. *Evaluating cross-platform mobile app performance with video-based measurements*. M. Sc. thesis, Faculty of Information Technology and Communication Sciences, Tampere University.

Dean A. Shepherd and Marc Gruber. 2021. The Lean Startup Framework: Closing the Academic–Practitioner Divide. *Entrepreneurship Theory and Practice* 45(5), 967–998.

Statcounter. 2021. Mobile Operating System Market Share Worldwide | Startcounter Global Stats. Referenced 10.3.2022. <https://gs.statcounter.com/os-market-share/mobile/worldwide/2021>

S.M. Sutton. 2000. The Role of Process in Software Start-Up. *IEEE Software*, 33–39.

Marko Taipale. 2010. Huitale – A Story of a Finnish Lean Startup. *Lean Enterprise Software and Systems* 65, 111–114.

John M. Wargo. 2020. *Learning Progressive Web Apps: Building Modern Web Apps Using Service Workers*. Addison-Wesley/Pearson Education, 2020.

Shane Warden. 2003. *Extreme programming pocket guide: team based software development*. O'Reilly, 2003.

Spyros Xanthopoulos and Stelios Xinogalos. 2013. A comparative analysis of cross-platform development approaches for mobile applications. In: *BCI '13: Proceedings of the 6th Balkan Conference in Informatics*, 213–220.

Tasnim Zohud, and Samer Zein. 2021. Cross-Platform Mobile App Development in Industry: A Multiple Case-Study. *International Journal of Computing* 20 1, 46–54.