jamk.fi

Using Google Ventures Design Sprint Framework for Software Product Development in Startups

Viktoriia Poliakova

Bachelor’s thesis
November 2017
School of Business
Degree Programme in International Business

jamk.fi

Description

Author(s) Poliakova, Viktoriia	Type of publication Bachelor's thesis	Date November 2017
		Language of publication: English
	Number of pages 71	Permission for web publi- cation: x

Title of publication
Using Google Ventures Design Sprint framework for software product development in startups

Degree programme
Degree Programme in International Business

Supervisor(s)
Saukkonen, Juha

Assigned by
Robo Technologies GmbH

Abstract

In the fast-paced environment of the startup world, companies are always trying to discover and execute the most efficient and effective ways of solving problems, developing new products and working in teams. Some frameworks that address this issue have been developed during the recent years, and they are actively used in various organizations.

The general aim of this study was to evaluate the effectiveness of such framework, Google Ventures Design Sprint, in a hardware startup called Robo Wunderkind. The objectives were to execute the Design Sprint in practice by developing a concept of a software product, analyze the results of the Design Sprint and evaluate its appropriateness for the company and its products.

The theory and knowledge base part of this study presents the general concepts of software development and its frameworks, design thinking and Design Sprint. This part further justifies the connection between these frameworks and their application in this research.

Action research was used as the research approach for this study, as it was similar to the cyclical nature of the Design Sprint framework and allowed deep reflection on the research process. The research consisted of five cycles that contained one action and one reflection process each.

As the result of this study, a concept of a software platform was created, the key challenges for its further development have been identified, and the appropriateness of the Design Sprint framework was assessed using action research.

The research findings can be utilized by researchers from different backgrounds in order to further study the Design Sprint framework and its application in various settings and fields.

Keywords/tags (subjects)
Google Ventures Design Sprint, design thinking, software development, innovation

Miscellaneous (Confidential information)

Appendices 1, 2, 3 and 4 (35 pages) are confidential until 12.11.2027.

Contents

Figures

Tables

The list of tables of this study is confidential until 12.11.2027.

Images

The list of images of this study is confidential until 12.11.2027.

1 Introduction

In a business setting, regardless of a field or an industry, it can be often observed that usual working methods and ways of solving issues stop to be effective and efficient, and simply do not bring outcomes. This observation can be applied to individuals in the work environment, as well as to whole teams and subteams inside of organizations of different types, be it corporations or startups. If something is not working, then certain steps towards the optimization and the ways to finding solutions to such sudden lags of productivity need to be sought. How such problems are solved, is solely up to individuals or teams, and at the end of the day, various ways of improving working processes are found. However, what if one specific framework could successfully address all similar issues, improve work processes and help individuals and teams to innovate, regardless of the environment they are in?

This study examines the effectiveness of such framework - Google Ventures Design Sprint (hereinafter referred to as GV Design Sprint, Design Sprint or Sprint) in a context of one specific company and a specific issue that the company needed to solve. This research is composed of 6 parts: Introduction, Research design, Theory and knowledge base, Results, Conclusions, and Discussion. Introduction part includes the background information to the study, description of the assignor company and research problem and questions of this thesis. Research design part elaborates on the chosen research approach and method and justifies their appropriateness for this study. The Theory and knowledge base chapter creates an academic background for the research by explaining the basics of software development and its methodologies. This chapter also explains the design thinking concept and elaborates on the Design Sprint framework, on which the research is based. The Results chapter describes the research process that took place. The Conclusions part presents the outcomes of the research and the answers to the research questions. Finally, the Discussion chapter includes the discussion about reliability and validity of this study, its limitations and suggestions for the future research on this topic.

1.1 Background and company information

The assignor for this thesis is the startup company called Robo Wunderkind, which is based in Vienna, Austria and Shenzhen, China. The company is a hardware startup in the field of educational technology: it produces modular programmable robotics kits for children. Robo Wunderkind was founded in 2013 with the mission of bringing robotics and coding closer to children, and making it simple and entertaining for them. In 2015 the company has successfully completed a Kickstarter campaign, raising $246,000 from backers coming from 58 countries, subsequently winning a number of awards and getting a vast media coverage worldwide. At the moment of writing this thesis, the company is in the pre-order phase and has three hardware products in its range, as well as two accompanying software products.

The author of this thesis has been a part of Robo Wunderkind’s team since March 2017 as an intern, and subsequently as a full-time employee in the marketing team. The motivation to write this thesis arose, firstly, from the author’s deep interest and dedication to the company and its products, and secondly from the interest in the GV Design Sprint framework itself, and the fact that it hasn’t been widely researched yet.

As a company, Robo Wunderkind operates in the extremely fast-changing environment of the technology business, thus the startup is constantly searching for new ways of boosting productivity, improving internal processes, and developing the best products and solutions for its customers. Moroni, Arruda, and Araujo (2015) believe that, in order to do this, many companies introduce some aspects of design to their innovation processes, thus making them design-driven. Such design-driven mindset allows to look at the same context and environment from a completely different angle, create new products and services with new meanings, and understand the target customers and their problems better. (ibid., 2200.) Prior to this study, Robo Wunderkind has already employed some aspects of design and innovation-driven thinking in the company’s marketing team, as well as has worked with some modern agile methods for rapid and innovative software development purposes in the technical team. However, it was not the case that the whole team has worked together on a specific problem within one specific framework for innovation, simply due to the fact that the engineering, sales and marketing, product and design teams at Robo Wunderkind are naturally often quite separated because of the essence of their tasks (although, of course, they still work together as one big team). That said, the company wanted to find a new innovative way to work effectively and efficiently altogether on specific problems, and the GV Design Sprint became such solution for Robo Wunderkind due to the reasons explained in the next subchapter.

1.2 Research problem and research questions

This study aims to research the GV Design Sprint framework in the context of developing a software product in a hardware startup. The background of the research problem lies in the following: Robo Wunderkind team members had an idea to develop a new software product, which would be independent of the company's main hardware product. The idea to utilize the Design Sprint framework for developing the product idea has been proposed, as the founders of the company have heard of the case studies where it was used for similar purposes of product development, and thus were enthusiastic to try it out at Robo Wunderkind. Therefore, the following research problem has been formulated by the author:

"How could the Google Ventures Design Sprint methodology be used for developing a software product in a hardware startup?"

The following research questions have then been derived out of the whole research problem:

1. What kind of software product, which would be independent of the company's main hardware product, could be created with the help of Design Sprint framework?

2. What are the product-related challenges that could arise after the product launch?

3. How well does the Design Sprint method apply to the development of a software product in a hardware company?

In order to answer these questions, the real Design Sprint process has been implemented in the company, in which the author of this thesis has also participated alongside the other team members. Action research has been chosen as the research method for this study, as described and justified in the next chapter.

2 Research design

2.1 Research approach selection

According to Creswell (2014), research approaches are the general series of steps and ways of undertaking a research project in order to narrow the researcher’s broad assumptions down to concrete methods of collecting, analyzing and interpreting the data. The decision of which research approach to undertake is mainly based on the researcher’s assumptions, research design, the methods of collecting, analyzing and interpreting the data, which apply to this specific research and on the nature of the research problem itself. (ibid, 3)

There are two basic approaches to conducting a research: quantitative approach and qualitative approach. Quantitative research applies to a phenomenon that needs to be researched in terms of its quantity, thus, yielding a concrete amount as the research result. Conversely, qualitative research aims to answer research questions regarding a qualitative phenomenon through assessment and analysis of underlying motives, attitudes, opinions, and behavior. The results of the qualitative research always appear in the non-quantitative form, or in a form that can not be analyzed quantitatively (Kothari 2004, 3-5).

As answering the research questions of this study does not yield any numerical results, but instead requires verbal interaction in a group of people, the quantitative approach would not have been suitable for this thesis, thus qualitative research approach was chosen. In addition, this approach was deeply rooted in the research method, which the author planned to select, so the decision on selection of the research approach for this study came rather quickly.

2.2 Research method selection - Action research

Since answering the research questions of this thesis implied developing, prototyping and testing an actual software tool in a team, the action research method has been chosen as a primary method to complete the research.

According to Stringer (2014), action research is a systematic and holistic approach to a research, which allows its participants to collaboratively find solutions to their particular problems. Action research involves and engages the rapid dynamics of any social setting, as opposed to other types of research, where general causalities, correlations, and explanations regarding a small number of things that are researched, are sought. An integral part of action research is continuous cycles of investigation that show the solutions to presented issues and suggest the possible means for participants to improve their work and to make sustainable developments in their actions and practices. The nature of action research relies on the statement that generalized solutions of plans in any social setting can be irrelevant and inapplicable to that specific setting or to groups in it, therefore, they should be modified and adapted to the environment they are used in. In a nutshell, the main purpose of action research is to involve all people who are influenced by the investigated issue, or, conversely, influence it themselves, in the process of systematic investigation and experimentation in order to achieve a certain goal and reflect on it. (ibid., 1-6.)

McNiff and Whitehead (2002) characterize action research as a special way to investigate one’s learning by looking at one’s practice in order to understand if it’s done correctly, reflect on it, and make changes and improvements if needed. This kind of research can be conducted in a variety of environments, such as social sciences, business management, education, and many other. Regardless of a context, the learnings in action research always come from action and reflection by participants. (ibid., 15-16.)

Zuber-Skeritt (2001) argues that the action research process can be broken down to four phases: (1) strategic planning, (2) acting (implementing the plan), (3) observing and evaluating, (4) critically reflecting on the results. Altogether, these phases represent a cycle, which, in an action research, continuously repeats itself (given that the reflection was utilized and the action plan has been revised), until the solution to a problem is found and/or the objective is reached. (ibid., 19-20.) McNiff and Whitehead (2002, 16), provide a similar explanation of these phases, breaking them down to gathering data, reflecting on an action, generating and validating evidence from the collected data, and making conclusions out of it. Below is the visual representation of action research phases as per Zuber-Skeritt (2001, 20), which suggests perceiving it as a continuous spiral of planning, where each cycle leads to a better understanding of an issue and brings participants closer to a solution.

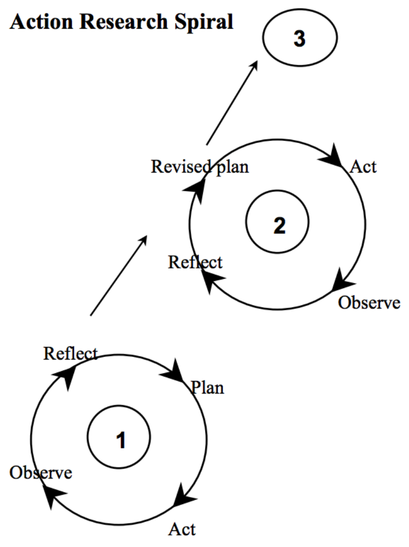

Figure 1. The spiral of action research cycles (Zuber-Skeritt 2001, 20)

Why action research?

The decision to choose action research method for this study was made due to the several reasons. First and foremost, the topic of the thesis itself was taken into the account. At the beginning, during the preparations for this study at the stage of deciding the topic, it was acknowledged that researching the GV Design Sprint framework in a real-life setting simply calls for action research, due to the fact that this framework itself represents a philosophy similar to the one of action research. Design Sprint method is based on collaboration, communication and knowledge sharing among the Sprint participants, its cornerstone is "learning by doing" principle, which is the same approach that action research represents. In addition, Design Sprint framework is cyclical too, which deemed to make researching it via action research phases and cycles reasonable. Finally, according to McNiff and Whitehead (2002, 85), action research is always practical and requires a deep focus of participants on one particular issue in order to progress in understanding it better, which is why it suited for this study where a group of people was challenged to develop a concrete product in a real-life practical setting.

Research process implementation

The research process has followed the pre-defined step-by-step framework of the Design Sprint method (discussed in more detail in Chapter 3). In addition, to meet the demands of action research method, cycles of action and reflection have been incorporated into the Design Sprint process. As the result, the research had five cycles that lasted one day each and had one action and one reflection process per cycle. The 5-day research process that took place is illustrated below.

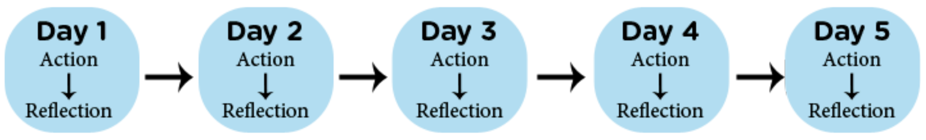

Figure 2. The implemented research process of the study.

After the data has been collected and analyzed, the conclusions of this study have been made and presented to the assignor company.

2.3 Data collection and analysis

In order to address the research problem and answer the research questions of the thesis, this study has followed the cyclical nature of the action research method, illustrated previously. The data for this research has been collected through observations, as the author has been directly involved in the action research process that has been executed via the Design Sprint framework. According to Kothari (2004), in the observation data collection method, the researcher extracts the information by observing the environment and not asking the respondents directly. When this method is used, the researcher should address the questions like "What should I observe?", "How should the observations be documented?", "How to ensure the validity of the results?". Overall, observation can be used as a method when it has a defined research purpose, when it is planned and documented in a systematic way, and checked for its reliability and validity. (ibid., 96.) As stated by Yin, doing a research by observing is a unique way of data collection, since the collected data goes through the own perceptions and feelings of the researcher, and thus it is not in any way altered or biased by what research participants are reporting. Such research can be both completely passive and participatory, but in both cases, the primary data is very valuable, as it is not filtered by the others. In addition, for the researcher, it is very important to decide correctly, when, where and what to observe. In qualitative research, researchers usually operate in a changing environment, and it is thus important to establish the clear vision of how to conduct an observation. When deciding on what to observe, many things can potentially be a subject for observation, namely characteristics of individuals, interactions between individuals, surrounding environments and actions that are happening around, both human and technical (non-human). (ibid., 143-145.)

Although the author of this study has participated in the process of execution of the Design Sprint, the results of this thesis came from the author's observations of the team behavior, the opinions that the team members have expressed, and from the whole process of executing the Design Sprint framework in practice. The collected data has been analysed through the reflection processes that took place at the end of each of the five cycles during the research process, and the answers to the three research questions have been found at the various cycles of this research.

2.4 Plan for research ethics and quality

Research ethics

According to Aguinis and Henle (2004, 35), prior to starting a research project, one should assess his capabilities and proficiency to execute a research, his awareness about the ethical guidelines that exist, the correctness and appropriateness of the created researched design, and if the research is acceptable to bring to the life from the ethical perspective. Ethics in research exist for the sake of making sure that the research brings more benefits than harm to the research participants and the society. This is done, namely, by ensuring the informed consent of all involved parties regarding the nature and the purpose of the research, by minimizing any potential harm or risk for participants, treating all research participants with respect to their personas and their privacy, diminishing the chance of the waste of time for participants, etc. (Eriksson, & Kovalainen 2016, 63; Aguinis & Henle 2004, 36.)

In addition, researchers are responsible for ensuring the originality of their work and avoiding plagiarism issues at all costs (Eriksson, & Kovalainen 2016). Plagiarism might happen even unconsciously by forgetting to cite sources or by doing it in a not correct manner, thus breaking the ethical principles and potentially infringing the copyrights. Therefore, researchers should always give credit to other authors they refer to in their works, and be careful in citing the used sources correctly. (ibid., 75-76.)

The author of this study takes the ethical principles seriously and thus is aware of her moral responsibility towards all parties involved in this research. All research participants are treated with respect and are provided with full information about the contents of this thesis and its purpose. The participants have given their permissions to record their answers and to subsequently use them in this study; however, as per participants' request, their names are not mentioned and not disclosed to any 3d parties. The participants have also given their consent to use the materials that have been produced during the research process, in this thesis. No participants were harmed in any way during this study, and the research will benefit their organization and to the further exploration of the researched issue. Finally, all ideas and concepts that have not been developed by the author of this thesis, are properly cited, and the original authors are referred to.

Reliability and Validity

The author has relied on the existing academic literature on various topics when working on this study. The examples of the literature included course books, articles from various journals and other research works, found in the libraries and on the Google Scholar platform, which is a source of information used by many academics, thus, supposedly, a reliable one. This study can also be used in the future by other researchers who study the Design Sprint framework or any similar topic. The data collected in this study originate from using the Design Sprint framework in practice, and thus can presumably be considered reliable, as this research is practical and observes the issue in the real-life setting. Moreover, the author of this study has planned the research process in a way that the finding directly answer the stated research questions and give a profound background to all of the answers, thus the validity of the research is planned in detail and ensured. The reflection on the planned reliability and validity of this study can be found in Chapter 6.

3 Theory and knowledge base

3.1 Software engineering

Software engineering, as defined by Pressman (2005), is a process, accompanied by a selection of tools and procedures that are used to build computer software. The objective of software engineering is to create customer-oriented software systems that are operating with efficiency, are tenable and sustainable. (ibid., 2-20.) Furthermore, such systems should be designed so that they can be successfully implemented in the set project time frames and budgets (Braude & Bernstein 2016, 1). Every software engineering project begins, to some extent, from a business need: it could be a need to improve and widen the functions of the existing software, update it according to the changing needs of the environment, or the need to create a completely new software product or system completely from the scratch. Developing a software product is a process of repeated learning, which yields something that can be named as “software capital” - an array of collected, distributed and organized knowledge and learnings during the implementation of the whole software development process. In addition, software engineering shouldn’t be confused with software development process. While both have to do with software development, a software development process can be generalized as a general methodology and philosophy to development, while the development (engineering) itself, in addition, refers to the technical methods being deployed (Pressman 2005, 15-20).

Software engineering is a complex and a challenging process, which requires cohesion between a number of people formed into interconnected teams, whose goals should be aligned around a specific product with a pre-defined framework to build this product. All this system should be then organized into one project with a step-by-step defined process of its execution. Altogether, the activities needed to be performed during software development are frequently referred as to “4 P’s of software engineering”, which are: people, product, project, process. “People” refers to all project stakeholders, in other words - participants of a project, and those who influence it. “Product” is something that is being developed, its subsequent end result and the documentation regarding it. “Project” means all the activities implemented in order to achieve the result and get to the end product. Finally, “Process” is characterized as a shared set of procedures within which the stakeholders execute the project. If all of these elements are balanced, properly taken care of and addressed, a software project is deemed to be successful, therefore, stakeholders need to make sure that 4 P’s do not potentially conflict with each other. (Braude & Bernstein 2016, 5-6)

Since this thesis describes the development of a software product in a company through the Design Sprint methodology (hence, the specific process), the Process element of the 4 P’s of software engineering is broken down in more detail below. This is done in order to build a big picture around the software development basics, subsequently address the Design Sprint framework and add more clarity to the main issue of this research.

3.2 Software process

As defined by Braude and Bernstein (2016), a software process is a framework for implementing the activities needed for a project in a systematic and organized way. Following a software process helps to reach a cohesion in a team and guide it through the tasks; it shows how different phases of a software project are interconnected and helps to define and reach the project's output. (ibid., 9.) Pressman (2005), summarizes the definition of software process as a "series of predictable steps - a road map that helps you to create a timely, high-quality result". He further claims that the individuals that follow software processes are normally software engineers and their managers, as well as those who have requested software out of their certain business needs. (ibid., 20.) In addition, according to Sánchez-Gordón and O'Connor (2016, 2), Zahran (1998) states that a software process is a series of practices and methods that are utilized in order to develop, sustain and preserve software and all documentation related to it.

While the aforementioned definitions vary in wording and there's no universal way to refer to software process yet, still, a similar pattern can be seen among them. Therefore, the following conclusion can be made out of it: any software process is a systematic and detailed way that helps a project's stakeholders to define the goals of a project, a step-by-step way to achieve them, and subsequently to manage and sustain the end result. Nevertheless, according to Sánchez-Gordón and O'Connor (2016, 3), Pressman (2009) believes that software process still doesn't provide a definite right or wrong way of approaching software development, but rather presents a flexible and adaptable framework that lets teams choose and act upon the appropriate course of action and tasks. Sommerville (2011, 28) also suggests that an ideal software process doesn't exist as it vastly depends on people who are making decisions and judgments and thus adapts to the capabilities and resources of stakeholders, and the specifics of the projects.

3.3 Software development methodologies

As stated by Sommerville (2011), software processes can fall into the category of either plan-driven or agile processes. Plan-driven processes can be defined as those processes, where the actions are planned beforehand and the measurement of progress is based according to this initial plan. Agile processes, on the other hand, suggest that planning is adaptive, and thus can be done on the go as the project evolves and the needs of the customers are changing. (ibid., 29.) Each of these two paradigms represents many different software development methodologies, and the most significant ones are further discussed in this subchapter.

Plan-driven methodologies

Boehm's 1988 study (cited in Gill 2014, 1), suggests that the plan-driven methodologies have been introduced in the past in order to manage large and important projects using fixed and repeatable processes. According to Petersen and Wohlin (2010, 657), Hirsch (2005), defines the following characteristics of the plan-driven approach: the functionality of the software should be defined before the start of a project, a detailed step-by-step plan of a project is required, all requirements are very detailed and if something needs to be changed, it is implemented after the output is produced. Moreover, the architecture and design elements must be specified before the actual development starts, coding happens only during one specific stage, testing happens at the end of the project, and quality control process is formal. Plan-driven methodologies are considered traditional, and they have been further defined as "heavyweight" due to a very elaborate set of requirements, exact documentation, planning and inspection (Awad 2005, 1). There are many plan-driven methodologies that are practiced, and this study will look at the two of them: the Waterfall model and the Spiral model.

The Waterfall model

Munassar and Govardhan (2010) claim that the waterfall model is considered classical and one of the oldest among the other models of software engineering. The most common instances where it's used are governmental and big companies' projects. (ibid., 95.) The basic principle of using waterfall model implies that one can start a new stage of a project only after completing the previous one, thus the phases are independent and should be performed in a sequence (Modi, Singh, & Chauhan 2017, 117-118). According to Kaur and Sengupta (2011, 1), the waterfall model has been frequently used in the past due to the very formal inspection requirements that it imposed. This model puts emphasis on the early-stage planning of a project in order to minimize potential mistakes, as well as on the detailed documentation and planning, which makes it a good fit for those projects, where the quality concern is of a very big importance (Munassar & Govardhan 2010, 95). Ruparelia (2010) believes that waterfall model performs the best when creating big software projects that serve as back-end functional to smaller software projects. For instance, an example of such software could be secure operating systems. (ibid., 8-9.)

As noted by Munassar and Govardhan (2010, 95), a typical waterfall lifecycle consists of a sequence of steps, which begins from specification and establishment of system and software requirements, proceeding to modeling and design, then to actual programming, to verification and testing, and finally to the support and maintenance activities. The figure below illustrates this sequential model.

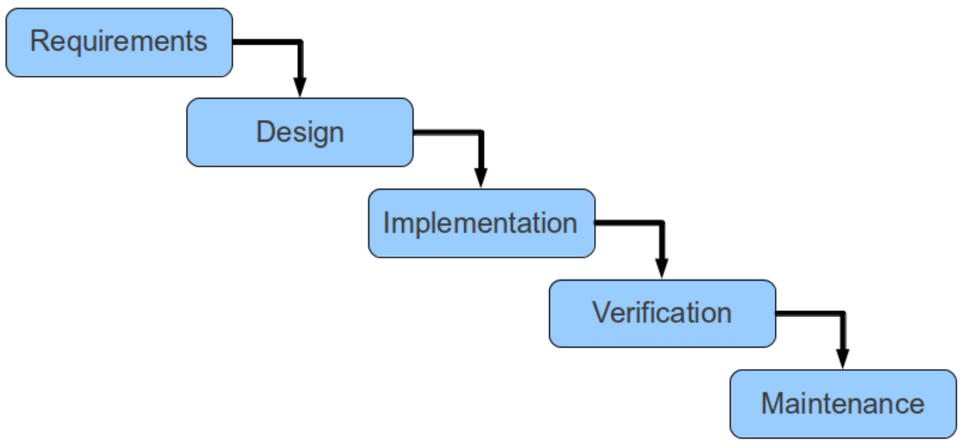

Figure 3. The Waterfall model of software development (SpriteCloud n.d.)

The disadvantage of the waterfall model is the lack of feedback in between the stages, which often leads to problems and issues with the software only being discovered after the implementation of the whole project (Sommerville 1996). In addition, it is also difficult to make any changes, which can be requested by the customer/end user, after the final maintenance stage. This results in the final outcome often not meeting the customers' needs, which is why the waterfall model started to be widely criticized, which led to the further development of other software development models. (ibid., 269)

Spiral model

The spiral model was firstly developed by Barry Boehm as a result of various cases of adjustments to the classic waterfall model in big software projects (Awad 2005, 6). According to Pressman (2005, 54), this model combines prototyping of the desired end product with the very systematic and detailed approaches of the waterfall model. Boehm (1988, 61) himself defines the major feature of this model as the adoption of the risk-driven approach to software development, as opposed to traditional document-driven or code-driven processes. Fairbanks (2010) describes the risk-driven approach as the one where the efforts that the stakeholders put into the project are equivalent to the risks that the project faces. To put it simply, this approach advocates for well-thought-out allocation of resources, and it allows the project's stakeholders to identify and adequately address the project's risks in a timely manner. (ibid., 36.)

The basic idea of the spiral model implies that software is developed in cycles that evolve: during the early cycles, a prototype of the final outcome is developed; at the later stages, developers build up on the prototype and produce more sophisticated and complete versions of it (Pressman 2005, 54). In his later publication, Boehm (2000), defines the spiral model as a *"risk-driven process model generator",* which has two main features: cyclic recurrence in order to gradually reach the final outcome of a software project, and a number of so-called *"anchor point milestones"* that ensure the mutual understanding and commitment of the stakeholders. In this definition of the spiral model, risks are events that can trigger the failure of a project, and anchor point milestones refer to the means of progress tracking and comparison between different cycles in a spiral model. (ibid., 3-4.)

Below is the visual representation of the spiral model, which illustrates its cyclical nature.

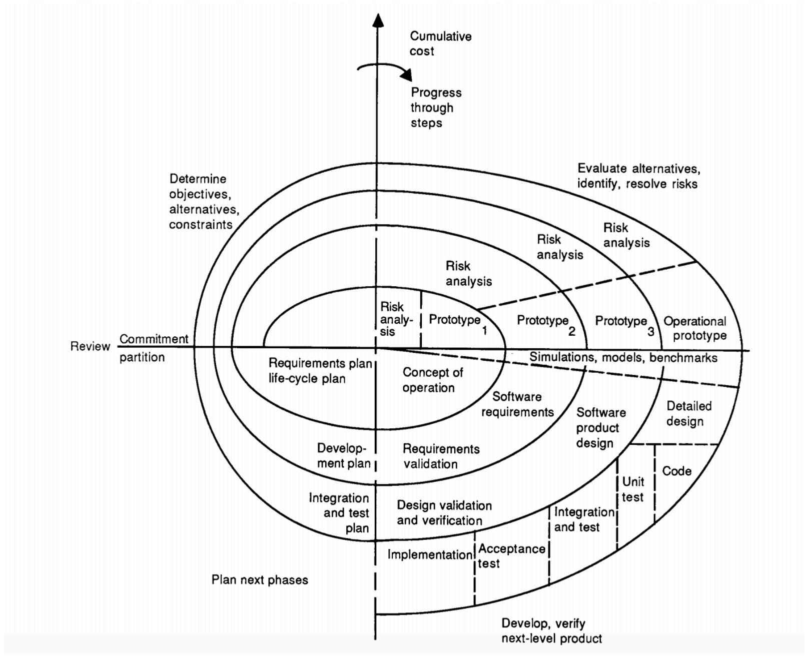

Figure 4. Spiral model of software development (Boehm 1988, 64)

As illustrated in Figure 3, the spiral model has four phases, which move across four quadrants in a repeatable cycle. According to Ruparelia (2010), the phases are:

1. Setting the objectives of a project.
2. Consideration of the alternative objectives, analysis of possible risks.
3. Verification, development, and testing.
4. Planning of the next cycle. (ibid., 10-11.)

As the phases progress and improve, a prototype of an end result is created along the way, in accordance with the requirements and testing procedures (Ruparelia 2010, 10). In addition, these phases and cycles of the spiral model are adaptable and applicable to any stage of a lifecycle of a software development project until it goes out of the use or simply gets outdated, unlike the other models which lifecycles end when the work on the project is complete (Pressman 2005, 55).

One of the main benefits of spiral model is its emphasis on the risks and costs of the project specifically from the beginning of it (Ruparelia 2010, 11). This risk control is the cornerstone of this model, and the further advantage is its requirement of risk evaluation and reduction during all stages of a project (Pressman 2005, 55-56). However, according to, Bernal & Karam (2016, 64), the drawback of the spiral model lies in the assumption that software developers are able to correctly identify and eliminate risks, which might not happen in all cases. In addition, this model requires a very flexible project management and strongly adaptable documentation processes between the stakeholders as the product prototype evolves in the spiral (Ruparelia 2010, 11). Finally, the spiral model can be simply very costly due to the fact that the resources needed for its implementation increase cycle by cycle, and the constant risk analysis is required all the time (Modi, Singh & Chauhan 2017, 118-119).

Agile methodologies

According to Braude and Bernstein (2016), agile methodologies have been developed in the 1990s as an alternative to the classic plan-driven methodologies that were seen as too plan- and requirements-concentrated. One of the biggest issues of such methodologies is an unclear set of requirements for the end product at the very beginning of the project, which causes many projects that follow such methodologies face major obstacles during the development. Agile methodologies address such issue by providing adaptable, efficient and responsive frameworks. The cornerstone of all agile methods lies in the Agile Manifesto - a summary of the key principles of agile methodologies that have evolved over time, which was developed in 2001 by the software industry experts. Agile Manifesto can be broken down into four points:

1. Interaction between individuals is put over processes and tools.
2. Well-developed and working software over heavy documentation.
3. Collaboration with customers over negotiating contracts and requirements.
4. Addressing changes over following particular plans. (ibid., 63-64.)

Abrahamsson, Salo, Ronkainen, and Warsta (2017) suggest that a development process is considered agile when it’s incremental, cooperative, straightforward and adaptive. Incremental implies rapid and small software release, cooperative means close cohesion and communication between the project’s stakeholders, including customers. Additionally, straightforward means that it is easy to learn and document the whole process; finally, adaptive refers to the ability of making changes to the project at any stage. The authors further specify that the main features of agile methodologies are simplicity and their fast speed, which allows the developers to prioritize the most important functions and issues first, develop them, test and collect feedback in order to further address it. (ibid., 17.) Overall, according to Braude and Bernstein (2016), agile methods suggest creating a set of viable guiding principles for a project’s stakeholders rather than specific rules that must be followed. These guiding principles are developed in order for the stakeholders to decide on the appropriate practices and solutions as the project evolves. In addition, agile methods value creative thinking while solving problems, as opposed to the plan-driven methodologies where adapting to the prescribed rules is expected - such practices are seen ineffective and dangerous in the agile methods. (ibid., 65.)

Similar to the big variety of the plan-driven methodologies, many agile methodologies have also been developed over time. This thesis will look at the two of them, which are commonly used: Extreme Programming (XP) and Scrum.

Extreme Programming methodology (XP)

According to Lindstrom and Jeffries (2004), the method of Extreme Programming is based on simplicity, open communication, and feedback between the project’s stakeholders. XP methodology is very customer-centered, and the other stakeholders, such as business, development and testing teams all work together and handle all aspects and issues of the project. In Extreme Programming, teams develop software in small batches in a short time and in a consistent manner, in order to track the progress, collect the feedback and decide on the further actions and to continue improving the software design. (ibid., 44-45.) Beck (1999), underlines the following features of XP:

1. Planning - programmers estimate the scope of work for a project and customers decide on the scope and the timing of the small releases that programmers produce.
2. Small releases - a software project is developed in a series of short releases, which are regularly updated and renewed.
3. Metaphor - customers and developers define the project by certain metaphors that they share with each other in order to have a common vision of how the system should work.
4. Simple design - the software solution is designed as simply as possible without any unnecessary complexities.
5. Tests - programmers constantly test the functionality.
6. Refactoring - the developed system is adapted on the way by transforming and evolving the design without changing the functionality of the software.
7. Pair programming - the coding process is done by two people on one computer.
8. Continuous integration - when a new piece of code is produced, it is integrated into the existing system immediately.
9. Collective ownership - any programmer from the team is able to improve any part of the program anytime if they see a need for it.
10. On-site customer: a customer works together with the developer team and is physically present with them.
11. 40-hour weeks: it is forbidden to work overtime two weeks in a row, otherwise this is seen as a problem that needs to be solved.
12. Open workspace: team works in a large open space.
13. Rules - although the rules for the project do exist, they are flexible and can be always adapted if all stakeholders agree on them. (ibid., 71.)

The lifecycle of a project that follows the XP methodology consists of 6 phases: Exploration, Planning, Iterations to release, Production, Maintenance and Death (Awad 2005, 9). Below is the visual representation of this lifecycle.

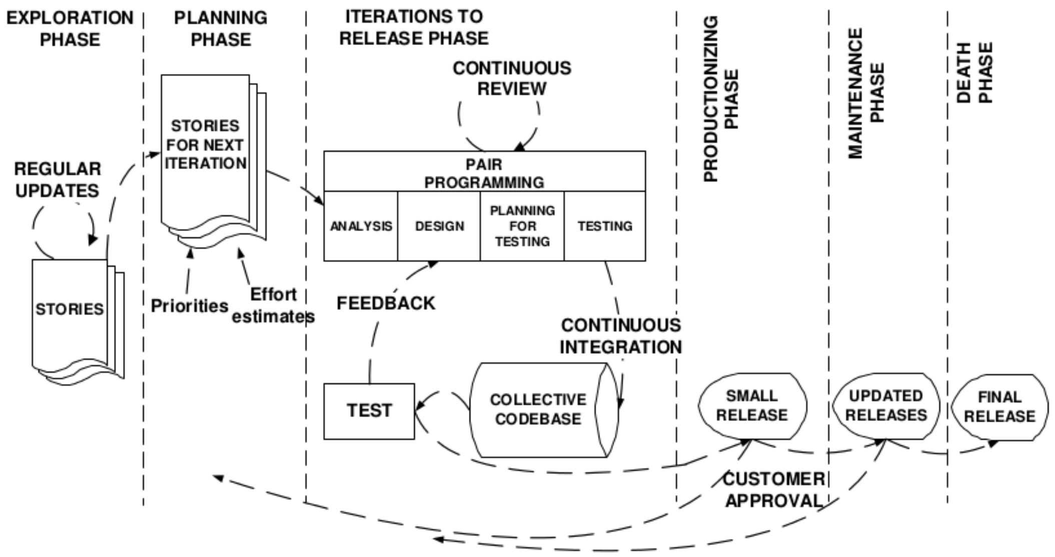

Figure 5. Extreme programming (XP) methodology of software development (Abrahamsson, Salo, Ronkainen, & Warsta 2002, 21)

Exploration phase requires customers to create story cards that communicate their vision of the end product and its desired functions (Awad 2005, 9). In the planning phase, the approximate timeframe and plan for the product releases are set (Lindstrom and Jeffries 2004). The Iterations to Release phase proceeds with the developing software in repetitive two-week cycles with the goal to produce a working software at the end of each iteration. The customer also contributes by communicating the desired features to be developed in the course of the next two weeks. (ibid., 47.) This phase further evolves into the Production phase, where extra testing and changes are done before the final outcome is released to the customer (Awad 2005). In the Maintenance phase, all development ideas that have not been implemented previously, are integrated into the system for the updated releases. The final Death phase occurs when the customer doesn’t have any more requests to be implemented and thus no further changes to the architecture of the whole system are made. (ibid., 9.)

Abrahamsson, Salo, Ronkainen, and Warsta (2017, 26) suggest that the Extreme Programming method delivers the best results when used in small and medium-sized teams. However, according to Lindstrom and Jeffries (2004), XP is often criticized because it is seen as too simple to develop beyond the stated criteria that some projects require. Thus, as the authors further claim, in order to evaluate the appropriateness of using XP in a project, one must take into the consideration the following: firstly, whether the team is able to communicate constantly and be cohesive and whether the team is comfortable with creating simple solutions. Moreover, one needs to evaluate whether the stable and constructive feedback system is set up and whether the team members are ready to show initiative and are not afraid of failure. If these questions are positively answered to a great degree, then the process of adoption XP practices happens much easier. (ibid., 43-50.)

Scrum methodology

Scrum is another widely used agile methodology that requires high levels of flexibility and adaptability. According to Abrahamsson, Salo, Ronkainen, and Warsta (2017), the basic principle of Scrum lies in the belief that development of any software is a complex process with many details to take into the account, such as resources, time frame, customer requirements, etc., and these details can naturally change during the development process. This is why development is complex and unpredictable, which calls for a high level of teams' adaptability in order to address the changes that are happening in the project. (ibid., 27-28.)

Schwaber (1995), suggests that Scrum consists of three groups of phases: Pregame, Game, and Postgame. The Pregame phase consists of the initial planning of the development process, which includes the overview of the software release, cost and timeline estimations. Additionally, this phase also includes the design and architecture of the software product. In the Game phase, the actual development is happening in a series of short "Sprints" - sets of development activities that are implemented in a limited time frame (usually 1-4 weeks) in a rapid manner. The Game phase normally consists of several sprints in order to intensify the development and constantly improve and adjust the system. (ibid., 125-126.) One of the crucial parts of the Game phase is daily team meetings, in which the following questions are addressed to each team member: 1) What have you done yesterday? 2) What will you do today? 3) Which challenges might prevent you from doing this?

(Ionel 2008). By answering these questions, team members are able to see each other's progress as the project evolves, they presumably become more eager to involve and contribute to the project and they stay more motivated. (ibid., 438.) Finally, the Postgame phase includes the final preparations before the release of the software, which includes final documentation, testing and the release itself (Schwaber 1995, 125-126). Below is the simple visual representation of Scrum process, which illustrates the three phases and how they are connected to each other.

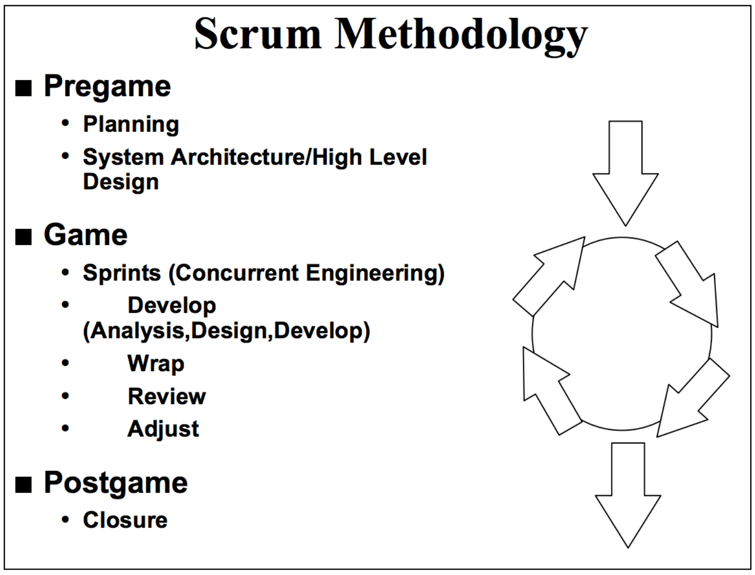
Scrum Methodology
■ Pregame
• Planning
• System Architecture/High Level Design
■ Game
• Sprints (Concurrent Engineering)
• Develop (Analysis,Design,Develop)
• Wrap
• Review
• Adjust
■ Postgame
• Closure

Figure 6. Scrum methodology of software development (Schwaber 1995, 126)

Scrum methodology is most efficient when used in small teams that consist up to 10 engineers (Abrahamsson, Salo, Ronkainen & Warsta 2017, 36). The biggest advantage of Scrum is its flexibility and adaptability to changing environments during a project that enable developers to learn on the go, collaborate between each other and create the most appropriate solutions for a project (Schwaber 1995, 129-130). However, according to Ionel (2008), a potential drawback in Scrum approach, ironically, lies in this collaboration and constant feedback loop: in order to successfully implement Scrum in a project, it is recommended that the main client constantly participates in the development activities by, for example, testing and providing feedback. That said, the client's availability is not always possible, especially when the client is external. Therefore, it is highly preferable to use Scrum in projects where clients are always available and open for collaboration (or come from the same organization and are internal), since this directly influences the project's outcome. (ibid., 439.)

3.4 Design Thinking

Since the Design Sprint method, which is the primary topic of this thesis, uses design thinking as a part of its general philosophy (to be discussed in more detail in the next chapter), the concept of design thinking needs to be broken down in order to give a clearer picture of what Design Sprint framework is based on.

According to Razzouk and Shute (2012, 330), design thinking is usually seen as an analytical and creative process that lets individuals who are undertaking it to create, experiment and test ideas, collect feedback about their projects and creations, and make changes to them based on it. This process can be seen as a special way of thinking and approaching a ground understanding of any new and emerging concepts, issues, events, etc., that leads to innovation and alteration in the way we live, manage businesses, people, etc (Tschimmel 2012). Even though initially, the concept of design thinking was based on the way designers and creative people approach work and problem-solving activities, nowadays, this concept is applied to any of such processes, regardless of a team or of an organization. Today's idea of design thinking became an effective solution for any kind of innovation process by merging creative thinking with the usual strategic business mindset, and thus it is very actively explored and used in various areas of different business functions, for instance, in management and marketing. (ibid., 1-2.)

According to Tschimmel (2012), the cornerstone of design thinking in solving problems, seeking solutions or creating new innovations is the design thinker's ability to combine and consider three following things simultaneously: needs of people, available resources, and opportunities and obstacles of the particular thing the design thinker is working on. If one utilizes design thinking approach, he should also be ready to combine analytical and emphatic thinking, be logical and emotional, follow methods and rules, but be flexible and spontaneous at the same time. Another typical feature of design thinking process is visualizing the ideas in the forms of sketches, drawings, notes, and prototypes - this helps to bring clarity to the whole issue being worked on, and helps design thinkers to find out further things that need to be worked on within this issue. Furthermore, an essential trait of design thinking is its human-centered approach, which means constant collaboration and communication between people in a team, and acting in a participatory way. Moreover, design thinkers are supposed to not only be working among themselves but also to involve customers, end users and other stakeholders of their projects in the design thinking process. Such cooperation improves the end result of the whole work, increases the effectiveness of reaching the outcome and improves the satisfaction of the future users. (ibid., 3-4.)

All in all, design thinking is a very specific approach to solving complex problems and/or implementing complex ideas, which requires multidisciplinary groups of people deploying user-centered and participatory approach (Thoring & Müller 2011). Design thinking found its usefulness not only in design fields but also in business and engineering areas in order to cultivate innovation. Nowadays, it is becoming more and more used and explored in many other areas for such purposes. (ibid., 1.)

3.5 Google Ventures Design Sprint as a new method of Design Thinking

The very first rough implementation of Design Sprint framework was initially made in 2009 by Jake Knapp, who at that time was an employee at Google (Knapp, Zeratsky, and Kowitz 2016). The idea of design sprints emerged out Knapp's goal to improve team processes at Google and to make the outcomes of usual team brainstorms better. Several years later, Knapp joined Google Ventures (later referred to as GV) - a venture capital created by Google that invests in startups. Google Ventures got interested in the idea of running design sprints with their portfolio startups, as these sprints could help young companies to test their ideas before launching their products. Jake Knapp was joined by Braden Kowitz, John Zeratsky, and Michael Margolis, and together they started implementing design sprints with GV portfolio startups, experimenting with the process, adjusting and improving the framework. (ibid., 1-5.)

According to Knapp, Zeratsky, and Kowitz (2016), Design Sprint is a five-day process that helps companies to answer crucial business questions and problems by creating prototypes and testing them with potential customers. In its essence, this process combines the ideas of business strategy, innovation, design, psychology and many more, that are combined altogether in a ready framework, which can be used by any team regardless of a background. Since the introduction of design sprints, they have been run not only by startups but also by investment bankers, engineering teams, and even school students, which suggests that this framework is applicable to various types of teams with different backgrounds and problems. (ibid., 6-16.)

There are three things that need to be taken care of before the beginning of a sprint (Knapp, Zeratsky and Kowitz 2016). Firstly, a specific challenge must be acknowledged and understood by the team. At GV, startups are encouraged to use design sprints for solving most critical problems in the company, because the idea of solving such extremely important issues makes team members be much more motivated and eager to solve them than it would have been in case of solving regular day-to-day questions. Secondly, a team for running a sprint should be formed. It is recommended that the team size should be not more than 7 people due to the fact that with larger numbers of participants it is difficult to keep everyone efficient and focused on work. What is more, teams should be cross-functional and mixed, and there should be an expert on specialized topics present during sprints. This requirement should be followed because having people from different backgrounds from the same company, working together in a sprint, can deliver valuable insights and help look at the issue from different perspectives. A “Decider” should be appointed - this person makes the final decisions during a sprint, and this is usually the company’s CEO/founder, or any other company’s important decision-maker. Additionally, a Facilitator must be chosen, who will guide the whole team through the sprint, manage time, discussions, and summarize the results. Finally, time and space for a sprint should be arranged. A team should shut down all operations for one working week and be present in the same room Monday to Friday from 10 a.m. to 5 p.m. (from 9 a.m. on Friday). Testing design sprints at GV proved the sprint duration of five days to be most efficient among all other options, such as ten days, one month, etc., because five days give a sense of urgency, but at the same time they give enough time and resources to get through a sprint and solve the challenge. Moreover, no devices are allowed to be used during the sprint (unless needed for a specific sprint's purpose), and the main prerequisite is that the whole team must be 100% concentrated on the challenge. (ibid., 21-42.)

The five-day design sprint process, developed by Google Ventures, is the following:

1. Monday - the long-term goal regarding a specific issue is chosen, and a map of the whole challenge about it is created in the team so that everyone has a shared vision of the problem Knapp, Zeratsky & Kowitz (2016). Then, the company's experts on specific topics share their knowledge and opinions, and a target of the sprint is chosen: what exactly needs to be solved in these five days, and what are the challenges involved. The whole process is documented on the whiteboard and on the sticky notes. (ibid., 51-88.)
2. Tuesday - on this day, solutions to the problem are created Knapp, Zeratsky & Kowitz (2016). Firstly, all team members conduct a research on existing ideas and case studies. Next, everyone comes up with their own ideas regarding a potential solution to the issue and anonymously sketches them on in a limited time frame. All sketched are collected and left untouched until the next day. (ibid., 93-118.)
3. Wednesday - team members give feedback and critique on each of the sketched solutions Knapp, Zeratsky & Kowitz (2016). Eventually, the solutions are voted for, and the solution to work with further is decided by the team's "Decider". The idea from the winning sketch is taken as a base for the prototype, and it is also enhanced by ideas from other sketches. Based on all these ideas, a storyboard for the prototype is created. (ibid., 125-158.)
4. Thursday - on this day, a realistic prototype is created using the Wednesday's storyboard Knapp, Zeratsky & Kowitz (2016). It can be created on screen, on paper, in a form of script with actors, in a form of a physical space, a 3D printed object, etc. (ibid., 163-190.)
5. Friday - the last day of the sprint is reserved for testing the created prototype by interviewing target customers and observing their reactions to the prototype Knapp, Zeratsky & Kowitz (2016). This allows the team to look at

their ideas through the customers' eyes and to show them the problems that can't be seen or predicted internally in a team. (ibid., 193-211.)

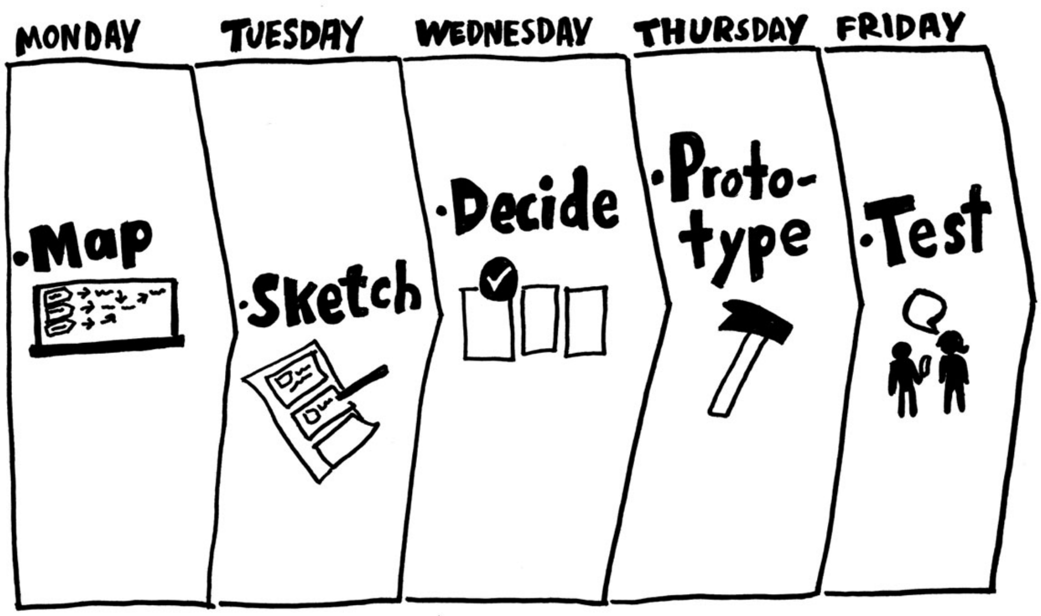

Figure 7. Google Ventures Design Sprint 5-day process (Tetuan Valley 2017)

According to Mucha and Nebe (2017), one distinguishing feature of GV design sprint is that it delivers very quick findings and results. In addition, sprint participants feel a sense of accomplishment due to the tangible prototype that they manage to create and test in five days. (ibid., 3.) Moreover, one further benefit of the design sprint framework is that it allows to rapidly create and test ideas without taking long periods of time to build and launch them, so that potential months of work shorten down to five days only (http://www.gv.com/, 2016). It can be thus summarized that design sprints help find a way not to only solve important problems, but to do it much faster, efficient and on a bigger scale (Knapp, Zeratsky & Kowitz, 2016). With the help of sprints, feasibility of new ideas can be assessed, existing products can be improved and business strategies can be created. (ibid., 6-16)

3.6 Synthesis of the conceptual framework

In the theoretical part of this study, the concepts related to software development have been introduced alongside the basics of design thinking and the detailed description of the Design Sprint framework as the ground concepts of this work. Since one of the questions of this study aimed at finding out what kind of a software product could be created through the Design Sprint, it was seen as necessary by the author to give an overview of how software products are usually developed, tested and brought to life. Moreover, while software development methodologies and design thinking are not directly connected to the Design Sprint method itself, and during the sprint, the actual programming or any other technical things happen quite rarely, still some important similarities between the three concepts can be seen, which are discussed below.

1. Sequentiality of the traditional software models and the Design Sprint

As discussed previously, the traditional software development frameworks tend to be very process-oriented and therefore emphasize a very detailed and thorough approach to the work that needs to be done. This can be seen from the Waterfall model and the Spiral model that have been presented: the first one has a sequence of steps, where one can move between the steps only having completed the previous steps; the latter is based on four phases that repeat themselves in a cyclical nature until the desired outcome is reached. Similarly to the structure of the traditional software models, the Design Sprint framework also follows the concrete series of steps that strictly go one after another. In this framework, it is impossible and senseless to start, for example, from the Step 2, which is solution sketching, without having identified the problem to be solved at the Step 1. It is also not possible to skip a step and to move forward without the outcomes that needed to be produced during it. Therefore, the nature of the Design Sprint methodology is partially similar to the one of traditional software models, since it emphasizes a step-by-step gradual process, which needs to be strictly followed.

2. Rapidness of Agile methods and the Design Sprint

As mentioned earlier, one of the core features of the agile methodologies is that the product development happens very rapidly, cooperatively and it is easy to document. In one of the described methods, Scrum, the idea of sprints is even presented as the core one: in this method, it is crucial to be rapid in the development activities, be efficient, and to produce good outcomes as quickly as possible. This corresponds to the Design Sprint framework, where the main idea is to produce a viable prototype in a limited amount of time (five days), and many of the actions during each of the five days are also put under certain time limits. In addition, this framework is also collaborative, similarly to the idea of the Design Sprint, where work is happening in the team. Finally, agile methodologies place not much emphasis to documentation, and there is no need in heavy documentation in the Design Sprint framework either, since all ideas are collected on a whiteboard, sticky notes and A4 pieces of paper only. Therefore, interestingly enough, a similarity can be again observed, this time between the agile methods and the Design Sprint framework (even though the Agile methodologies are completely different from the traditional ones, which as well have similarities with the Design Sprint).

3. Creative problem solving in design thinking and the Design Sprint

In the subchapter 3.4 of this study it has been discussed that design thinking is a creative and collaborative approach to innovation and problem solving. GV Design Sprint, in its essence, has the same roots: it also totally encourages creative thinking in the process of solving problems and developing ideas and products, and it emphasized collaborative efforts. The creative part of the Design Sprint can be seen, for instance, in the sketching and prototyping processes, where participants need to create visual and tangible solutions. Similarly, the same methods can be detected in the concept of design thinking. Finally, both in the design thinking and the Design Sprint frameworks, a big importance is also placed on strategic thinking and planning, as, usually, business models are discussed and strategic product decisions are made when using these frameworks.

To conclude, even though software development methodologies, design thinking, and GV Design Sprint are essentially different concepts, they, apparently, share a number of similar characteristics. While the Design Sprint method in the most cases doesn’t include a lot of coding processes, its nature can still be compared to the traditional and agile methodologies of the actual software development. At the same time, it is also comparable to the idea of design thinking, which falls in a completely different category from software development. It can be thus said that the Design Sprint framework has combined “the best from the two worlds”, meaning that it has the structure, logic, and rapidness of software development methods, and the creativity and strategic approach of design thinking.

4 Results

The results of this study are confidential until 12.11.2027. The confidentiality of this study is agreed between the thesis author, JAMK University of Applied Sciences, and Robo Technologies GmbH. The results of the research can be found in Appendix 1.

5 Conclusions

The conclusions of this study are confidential until 12.11.2027. The confidentiality of this study is agreed between the thesis author, JAMK University of Applied Sciences, and Robo Technologies GmbH. The conclusions of the research can be found in Appendix 2.

6 Discussion

The discussion of this study is confidential until 12.11.2027. The confidentiality of this study is agreed between the thesis author, JAMK University of Applied Sciences, and Robo Technologies GmbH. The discussion of the research can be found in Appendix 3.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. 2002. *Agile software development methods: Review and Analysis.* Technical Research Centre of Finland, VTT Publications.

Aguinis, H., Henle, C.A. 2004. Ethics in Research. In Rogelberg, S.G. (Eds.) *Handbook of Research Methods in Industrial and Organizational Psychology.* Oxford, Blackwell Publishing, 34-57.

Awad, M.A. 2005. Report. *A Comparison between Agile and Traditional Software Development Methodologies.* The University of Western Australia, School of Computer Science and software Engineering.

Beck, K. 1999. Embracing Change with Extreme Programming. *IEEE Computer, 32*, 70-77.

Boehm, B. 2000. Special report. *Spiral Development: Experience, Principles, and Refinements.* Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA.

Boehm, B.W. 1988. A Spiral Model of Software Development and Enhancement. *IEEE Computer, 21,* 61-72.

Braude, E.J., Bernstein, M.E. 2016. *Software Engineering: Modern Approaches.* 2nd ed. Waveland Press, Inc.

Carmines, E.G., Zeller, R.A. 1979. *Reliability and Validity Assessment.* SAGE Publications, Inc.

Creswell, J.W. 2014*. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, Inc.

Eriksson, P., Kovalainen, A. 2016. *Qualitative Methods in Business Research: A Practical Guide to Social Research.* SAGE Publications, Inc.

Fairbanks, G. 2010*. Just Enough Software Architecture: A Risk-Driven Approach.* Marshall & Brainerd.

Gill, A. Q. 2014. Hybrid Adaptive Software Development Capability: An Empirical Study. *Journal of Software, 9,* 2614-2621.

Golafshani, N. 2003. Understanding Reliability and Validity in Qualitative Research. *The Qualitative Report, 8*, 597-606.

Ionel, N. 2008. Critical analysys of the Scrum project management methodology. *Annals of the University of Oradea, Economic Science Series, 17*, 435–441.

Kaur, R., Sengupta, J. 2011. Software Process Models and Analysis on Failure of Software Development Projects. *International Journal of Scientific & Engineering Research, 2,* 1-4.

Kimberlin, C.L., Winterstein, A.G. 2008. Validity and reliability of measurement instruments used in research. *Am J Health Syst Pharm, 65*, 2276.

Knapp, J., Zeratsky, J., & Kowitz, B. 2016. *Sprint: How to Solve Big Problems and Test New Ideas in Just Five Days.* New York: Simon & Schuster Paperbacks.

Kothari, C.R. 2004. *Research Methodology: Methods and Techniques.* 2nd ed., Rev.ed. New Age International (P) Ltd., Publishers.

Stringer, E.T. 2014*. Action Research*. 4th ed. SAGE Publications, Inc.

McNiff, J., Whitehead, J. *Action Research: Principles and Practice*. 2nd ed. London: RoutledgeFalmer.

Lindstrom, L., Jeffries, R. 2004. Extreme Programming and Agile Software Development Methodologies. *Information Systems Management, 24*, 41–52.

LoBiondo-Wood, G., Haber, J. 2014 *Nursing Research: Methods and Critical Appraisal for Evidence-Based Practice.* 8th ed. Mosby, Missouri.

Modi, H.S., Singh, N.K., Chauhan, H,P. 2017. Comprehensive Analysis of Software Development Life Cycle Models*. International Research Journal of Engineering and Technology, 4,* 117-122.

Moroni, I., Arruda, A., Araujo, K. 2015. The design and technological innovation: how to understand the growth of startups companies in competitive business environment. *Procedia Manufacturing, 3*, 2199-2204.

Mucha, H., Nebe, K. 2017. Human-centered toolkit design. In *HCITools: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits*. Workshop at CHI 2017, Denver, USA.

Munassar, N.B.A., Govardhan, A. 2010. A Comparison Between Five Models Of Software Engineering. *International Journal of Computer Science Issues, 7,* 94-101.

Petersen, K., Wohlin, C. 2010. The effect of moving from a plan-driven to an incremental software development approach with agile practices. An industrial case study*. Empirical Software Engineering, 15*, 654–693.

Pressman, R.S. 2005. *Software Engineering: A Practitioner's Approach.* 6th ed. McGraw-Hill.

Razzouk, R., Shute, V. 2012. What Is Design Thinking and Why Is It Important? *Review of Educational Research, 82,* 330–348.

Ruparelia, N.B. 2010. Software Development Lifecycle Models. *ACM SIGSOFT Software Engineering Notes, 35,* 8-13.

Sánchez-Gordón M., O'Connor R.V. 2016. Understanding the gap between software process practices and actual practice in very small companies. *Software Quality Journal, 24*, 549-570.

Schwaber, K. 1995. Scrum Development Process. *OOPSLA '95 Workshop Proceedings, 117-134.*

Sommerville, I. 1996. Software Process Models. *ACM Computing Surveys, 28,* 269-271.

Sommerville, I. 2011. *Software Engineering.* 9th ed. Pearson Education, Inc.

SpriteCloud, n.d. *Waterfall Model of Software Development*. Page on SpriteCloud website. Accessed on 4th of July 2017. Retrieved from https://www.spritecloud.com/2011/06/software-lifecycle-consultancy/waterfall/

Tetuan Valley. 2017. *When to do a Design Sprint?* Page on Medium website. Accessed on 4th of October 2017. Retrieved from https://medium.com/tetuan-valley/when-to-do-a-design-sprint-88e1e3355f05

The Design Sprint. N.d. Page on Google Ventures website. Accessed on 22nd of July 2017. Retrieved from http://www.gv.com/sprint/

Thoring, K., Müller, R. M. 2011. Understanding Design Thinking: A Process Model based on Method Engineering. In *International Conference on Engineering and Product Design Education*, City University, London, UK.

Tschimmel, K. Conference Paper. Design Thinking as an effective Toolkit for Innovation. In *ISPIM Conference Proceedings*. Manchester: The International Society for Professional Innovation Management (ISPIM), 1-20.

Tsui, F., Karam, O., Bernal, B. 2016. *Essentials of Software Engineering.* 4th ed. Jones & Bartlett Learning.

Yin, R.K. 2011. *Qualitative Research from Start to Finish.* New York: The Guilford Press.

Zuber-Skerritt, O. 2001. *Action Learning and Action Research: Paradigm, Praxis and Programs*. In Sankara, S., Dick, B. and Passfield, R. (Eds), *Effective Change Management through Action Research and Action Learning: Concepts, Perspectives, Processes and Applications*. Southern Cross University Press, Lismore, Australia, 1-20.

Appendices