Adapted Agile SDLC for Small-Scale, Feature-Driven Projects and its Impact on Remote Working Environment

By

Qazi Muhammad Abubakar

Supervisor

Dr. Malik Muhammad Zaki Murtaza Khan

A thesis submitted to the faculty of Computer Software Department,

Military College of Signals, National University of Sciences and Technology, Islamabad, Pakistan, in partial fulfilment of the requirements for the degree of MS in Computer Science (Software) Engineering

OCTOBER, 2021

Abstract

A Software Development Life Cycle (SDLC) must be followed by anyone engaged in developing or maintaining a software product, in order for it to function correctly and be delivered on time. Traditional SDLCs, such as the waterfall model, can be regarded the ideal method for developing software for large scale and complex software projects when requirements churn is less frequent, but can offer a number of difficulties in the case of small-scale products, with a high possibility of changing requirements, since there is little room for adapting modifications after the requirements are agreed upon. Agile methodologies such as SCRUM and Extreme Programming (XP) are suitable for small-scale projects since they emphasize on an iterative approach to software development. However, depending on the nature of the projects and the working environment, the SDLC may need to be tweaked to meet the requirements of the people and stakeholders involved. Most of the agile models, though very successful among the software teams, comes with a significant number of practices which need the team agile team to be co-located and the members of the team need to be collaborating face-to-face in meetings, sprint reviews, client collaboration, retrospection meeting etc. Owing the fallout of the COVID-19 outbreak around the globe, the work modality saw a sudden shift from people working together under the same roof in an office to people working remotely from their homes. Such a shift in the work modality affected the practices used by agile teams who were used to building quality software in co-located environment and thus, a number of bottlenecks were found in the existing models. This research will provide an empirical study of a proposed agile model for the development of small-scale software products in an iterative and feature-based approach where the requirements are volatile and the time-to-release is critical in nature, based on SCRUM, XPP, and FDD. Compared to previous models, the suggested approach helped finish projects within 10-15% of the anticipated timeframes. On the other hand, we will examine some of the issues that occurred when a sudden shift in work modality occurred during the COVID-19 outbreak and the challenges that were encountered while implementing existing agile practices, and we will discuss our proposed solutions for overcoming those challenges.

Key Words: *Agile Software Development, Modified Agile Model, Small-Scale Feature Driven Softwares, Quality Control Process, Distributed Agile, COVID-19*

Acknowledgements

Throughout the writing of my dissertation, I got a lot of help and encouragement. First and foremost, I would like to thank Allah Almighty for blessing me with the strength and knowledge to complete my thesis as I would never have been able to accomplish anything if it weren't for the Almighty's guidance.

I'd sincerely want to express my gratitude to Dr. Zaki Murtaza, who was instrumental in developing the study topics and methods. Your informative comments encouraged me to improve my thoughts and raise the quality of my work.

I would also like to thank my thesis Co-supervisor Dr. Saddaf Rubab for her excellent cooperation throughout my research.

I'd want to express my gratitude my thesis committee members Dr. Yawar Abbas and A/P Bilal Rauf who were always there to help me and guide me, whenever I needed it.

In addition, I'd want to express my gratitude to my parents and my siblings for their sound advice and sympathetic ear. You are always willing to help me. Finally, without the help of my friends Qamar Khurshid and Mohsin Abid, I would not have been able to finish my thesis. They offered interesting conversations as well as enjoyable diversions from my study.

DEDICATION

Dedicated to my parents and my late grandfather Qazi Muhammad Bashir whose guidance and exceptional cooperation led me to great accomplishments in my life.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1

1. INTRODUCTION

1.1 Chapter Outline

This chapter will be focused on providing the outline for this research. We'll take a quick look at why it's important to have a good engineering process in place for software development. Later on, we'll talk about why this study is being done and what research objective it is aiming to answer.

1.2 Software Development Life Cycle

There was a need for a defined procedure to be in place, from the beginning of the field of software engineering, to assist software engineers all over the world in performing their creative job. We initially saw flashes of an SDLC in the early 1970s, which subsequently became to be known as the Waterfall model [1] However, despite the model to be a groundbreaking event in the field of software engineering, it wasn't free of faults and even Winston felt compelled to bring out some of the model's faults, describing it as "risky and invites failure" since the testing procedure could only be carried out at a later stage and there was little to no space to go back into the earlier stages of the SDLC. Many additional models were developed in the following years in an attempt to make the software development process more flexible and less prone to failures.

During the last decade of the 20th century, we were introduced to the Rapid Application Development (RAD) [2], which was later followed by other models such as SCRUM [3] and XP [4], where the focus was on achieving a process that was iterative in nature and flexible to handle unclear and rapidly changing requirements during the later stages of the software development process. After the release of the Agile manifesto [5], all of these methods were known as Agile methodologies together.

1.2.1 Traditional vs Agile Software Development

In today's world of rapidly changing technology, it's very difficult to lock down a software project's requirements from the outset, since technology is always evolving. As a result, any software development process must be adaptable to changing requirements, and there must be a system in place where the software can be developed iteratively, with requirements broken down into user stories and a system in place to integrate user and stakeholder feedback as more features are added to the software.

Agile methods are an obvious and clear option for iterative, feature-based software development where the changing requirements are welcomed and accepted, according to the Agile Manifesto [5]. However, these models provide you with a set of processes and techniques that can help you get started with adapting agile methodologies for creating software product; depending on the nature of the software projects you want to build, we can draw inspiration from existing models such as SCRUM, XP, FDD, and others to create an adapted agile model that best meets our needs.

1.2.2 Software Quality Assurance

It's one thing to turn an idea into reality; it's another to get the implementation of that idea, in the form of a software, to be approved by the customer and the end user once they are pleased with the product's quality. The software quality assurance (QA) process is critical to the success of any software project, and it includes both software testing and user input integration after the product is deployed in its intended environment. It ensures that the software meets the anticipated level of quality.

1.2.3 Remote Working Environment and Agile Methodologies

The agile manifesto [5] puts on a great emphasis on face to face collaborations between the members of a co-located team for working efficiently in an agile environment. The agile methodologies are implemented in a number of agile models like Scrum, Extreme Programming (XP) and Feature Driven Development (FDD) [3][4][16] where different models cater for project of different sizes and varying complexity and each model comes with its own sets of practices and rehearses.

These agile models are adapted by a number of software teams working around the world but at the core of most of the practices involved in these models, there is a need for a co-located team meaning if we have implemented this model in our team then we're bound to have all members of the team working and collaborating under the same roof.

Following the COVID-19 outbreak around the world, in the early 2020, the work modality saw a major shift where most of the workers were asked to stay at their houses, to work and collaborate remotely with other team members, in an effort to stop the spreading of the virus through human contact. Though this step was necessary in making sure that all the members of the team stay safe, it posed a major problem for organizations working in an agile environment that required daily face-to-face collaboration within the members of the team.

1.3 Research Objectives

This research will be a study on the modifications made to the existing software life cycles and processes for small scale, feature driven software projects and how they benefited us in terms of time, development cost, quality of the software, client and end user satisfaction and how the modified processes helped in catering for a shift in work modality to remote working environment. The detailed objectives are to discuss:

- How Software SDLC was modified for fast feature driven development of small/medium scale Android projects.
- Identify how the testing process was carried out during the SDLC when the primary focus was on a working release, in a 3 weeks sprint.
- How the quality control process and user feedback was integrated - post the release.
- What major challenges were faced when suddenly shifting the work modality from in-house to remote work. How it affected our existing processes and what changes we made to existing SDLC and processes.

1.4 Research Motivation

This study will prove to be very beneficial for small scale organizations that are trying to establish a footprint in the software engineering arena by developing small scale working products or software tools for specific platforms such as android mobile applications. Those organizations can take help from this study and take it as a reference for modifying their existing software process in a way that can help pace up the development of their products and can lead to greater client satisfaction. Another area that our research can help in, would be in addressing some of the key problems faced while working remotely, following the corona virus outbreak. This research can prove to be beneficial in helping the small scale organization by discussing the modifications made to the existing software life cycles and processes for small scale, feature driven software projects, and the benefits provided in terms of time, development cost, quality of the software, client and end user satisfaction and the ability of the modified process in addressing a shift in work modality to remote working environment.

1.5 Thesis Structure

This thesis consists of following chapters:

- **Chapter 1:** Introduction to the topic, Research Motivation, Research Questions and Objectives are mentioned.
- **Chapter 2:** Provides the research background and literature review.
- **Chapter 3:** Delves into the nitty gritty details of the proposed model.
- **Chapter 4:** Discusses the challenges and proposed solutions for agile software development in a remote working environment.
- **Chapter 5:** Analysis of the proposed model and solutions for remote working environment.
- **Chapter 6:** Provide the overall conclusion along with the potential future works.

Chapter 2

2. LITERATURE REVIEW AND BACKGROUND

This chapter will be focused on providing background for this thesis as well as the current state of study on the research objectives that this thesis aims to answer. We'll take a quick look at the different SDLCs that are often used for software development across the globe, as well as their challenges and advantages. We'll also talk about the significance of having a strong software quality assurance and control process in place, and how agile methodologies in a remote working environment may assist overcome some of the difficulties that software engineers encountered when the COVID epidemic broke out across the world.

2.1 Software Engineering Process

Software Engineering stands out among the various engineering disciplines because the product created as a result of a software engineering process is intangible. It's a concept represented by binary 1s and 0s, and when properly programmed into a project, it may be very useful in addressing real-world issues and even serve as a catalyst for ground-breaking events like the moon landing in July, 1969. However, because of the unique and intangible character of the product we are creating, as well as the fact that we cannot ‘see' it with the naked eye, strong engineering and management ideas are required in order to get some insight into the process and enhance its efficiency.

In the field of software engineering, there are two kinds of software development SDLCs: one is completely sequential, such as the conventional waterfall model, and the other is strictly iterative, such as Agile models. Each step of the conventional SDLC is run independently of the others, and the result of one phase becomes the input of the next. In Agile models, the idea is to iteratively build a complicated software product in short 2 to 3 weeks sprints that are repeated until all of the requirements are incorporated into the software. Some features or priority needs from a larger list of requirements are created, tested, and incorporated into the current system at the conclusion of each of these sprints. We'll go through both of these methods in more depth, in the coming sections.

2.2 Traditional Software Development

During the early phases of the software engineering area, mathematicians were believed to be the major creators of software products, and the phrase software engineering was seen as irrelevant. [6]. The software development domain continued to become more complex as the system evolved and technology became more sophisticated, and the humans were able to develop more powerful hardware. The creation of software thus became more complex and there was a need for an engineering system that could take on the challenge and define the various steps involved in the development of a software product.

2.2.1 Waterfall Model

We now have a variety of SDLCs to select from, starting with the earliest, the Waterfall model [1]. The waterfall model, which was first used in the early 1970s, is regarded as one of the most famous traditional SDLCs. This model was widely accepted for the development of the software products around the globe and it was very successful in developing some large-scale software products where the requirements churn was less and the requirements were clear at the beginning of the project.

However, even when the Waterfall model was first established, there were a number of areas that were seen to be hazardous when the requirements were subject to change. The numerous overheads associated with the waterfall model, such as comprehensive documentation for each development step, may be expensive and time consuming [7][8]. This approach also has an issue with late software testing since testing can only be done after the whole implementation is complete, and there is little to no space to go back to earlier stages. [9]

Softwares are created in a linear manner in conventional software development, such as the waterfall model, with each step performed independently of the others. One of the most basic challenges in this model is that for the project to be successful and accepted by the client, both the development team and the client must collaborate in understanding the project's requirements, documenting them, and ensuring that all stakeholders' and end users' needs are met by the documented requirements, since the project can only be tested once all the phases of the SDLC are completed.

What makes this approach particularly difficult is that, in most cases, when developing complex software systems, the requirements are not clear at the outset of the project, and it is only as the project progresses into the development phase that a better understanding of the requirements emerges, which can be due to the client or development team not having a clear understanding of the requirements.

Another reason for traditional software development failure is that the client's needs are not met until the end of the SDLC because the entire process is linear and sequential, and the product that is being developed will only be available to the client once all phases such as requirement, design, implementation, and testing are completed. [12].

2.3 Agile Software Development

Though traditional software development, gave the software developer a clear and standardized set of engineering practices to follow [1][7], the rigid linear structure and no way of moving back to a previous change if the requirement churn is experienced, made it very difficult to develop a complex software solution, using those very techniques and if the requirements were unclear at the start of the project then there was no way of accommodating change once the requirements were understood in a better way and worse if the requirements were not documented correctly, the development team could end up delivering a product which wasn't needed by the client, resulting in the loss of valuable time and effort.

Agile development helps us to solve most of the problems found commonly in the traditional software development and agile development is about "feedback and change" as stated in [34] and it is process which is non-linear in nature, and a complex software solution is developed in multiple iterations or sprints, where each sprint involves development, testing and close team collaboration.

Agile manifesto comes with a set of core values and principles for agile software development. The purpose of those agile values and principles is to have the members of the team work in close collaboration and to use simple techniques instead of complicated tools and processes, aiming to enhance communication within the team and to reduce overheads involved with traditional software development, like the need for excessive documentation at the start of the project. These overheads when reduced can result in speedy recovery of working software in a short amount of

time and the agile methodologies, because of these techniques have proven to be 10 time faster than that of the traditional software development.

Every iteration in agile, we are continuously planning, developing, delivering, and providing feedback. The first stages of agile typically comprise of a high-level strategy and establishing a fundamental knowledge of the requirements. Once we have a set of requirements and a high-level architectural design in place, we begin the development process in the form of a sprint, during which actual development and testing of that sprint's requirements occurs, and at the conclusion of that sprint, we perform system testing and then roll out a release, providing value to the customer. According to the agile manifesto [5], the customer is a member of the agile team, and the programmers and testers are able to receive immediate feedback after each sprint, which helps both the customer and the development team to better grasp the project's requirements. It also allows the development team to plan their next sprint in a way that takes care of the implementing the feedback given from the client, in an efficient way. SCRUM, XPP, and FDD, to name a few, are SDLC models that apply the agile method of creating software, and we'll go over each of them briefly in the next sections.

2.3.1 SCRUM

Scrum is a widely used Agile approach for software development that offers an iterative and incremental framework [13]. Scrum strives to offer a development process where each release includes the requirements of all stakeholders engaged, thanks to a closely-knit team led by a Scrum master. Scrum approach is built on sprints. It refers to a full iteration lasting about 2-4 weeks (subject to modification at the discretion of the Scrum master) during which a number of functional components/features are introduced to the project. Transparency, inspection, and adaptation are all ensured by Scrum. [13] Sprint planning determines the requirements, timelines, quality, resource allocation, and other aspects of a release, making the process and expectations clear. Daily 15-minute standup meetings ensure that the release is progressing as planned. Finally, sprint review handles adaptation as well as any necessary changes and fine-tuning. It includes a survey meeting at the conclusion of the sprint to emphasize the key events as well as customer feedback.

Scrum's incremental approach works well for almost all project sizes. The development teams' efficiency is boosted by thorough planning and specification of requirements. Furthermore, the

Scrum master's continuous check and balance over the process via daily Scrum meetings reduces the danger of the project deviating from its original objective and vision. Sprint retrospection enables teams to anticipate future problems by drawing on their past and current sprint experiences and devising a clear strategy to address similar issues in the next sprint [13]. The project owner is responsible for determining the priority of work in each sprint, as well as ensuring that the development teams have a clear understanding of the project's requirements. The incremental and iterative stages, as well as well-defined rules and regulations for each role, keep the teams focused and on track. Along with the numerous advantages of SCRUM, there are a few places where it falls short. For example, this approach can only be utilized in a co-located team, which means that all team members and roles, engaged in actively creating that product, must be working under the same roof.

2.3.2 Extreme Programming (XP)

Extreme programming is another widely used Agile model for smaller and skilled teams. It's used when the project's requirements aren't clear at the start and an incremental model is needed which is "Agile" enough to accommodate changing requirements at later stages of the project and can also start adding value to the customer as soon as the development process is completed. [14] The programmers and testers are the primary actors in this paradigm; incremental cycles are short, and the whole team works together in a cycle to create a set of requirements for that increment. The average team size is two to three people, including the client as a member of the team.

Continuous modification of the program's code, which is mostly accomplished via pair programming, Testing, Communication, Simplicity, Feedback, Courage, and Respect are the key features and ideals of extreme programming. [15] The ideal environment for the Extreme Programming model, similar to the SCRUM paradigm, would be a co-located and competent team, working together to gradually develop some particular project, with unknown needs at the start.

2.3.3 Feature Driven Development (FDD)

Another Agile approach that is iterative in nature and focused on team participation is feature driven development, in which a product's feature sets are classified into a set of features and those features are prioritized for the development process. [16] FDD assists in the production of concrete

and tangible results in each iteration by providing a set of features that are chosen or selected for that iteration prior to the start of the development activity.

This approach, on the other hand, is intended for larger organizations with top-down decision-making power and larger projects, as opposed to other agile models in which ownership is shared by the whole team rather than a small group of people. Another distinction between FDD and other Agile methods is their dependence on documentation. Following the first stage in the FDD model, "Developing an overarching model," four main papers are generated: 1) Project Goals 2) Consider using case diagrams. 3) Requirements that are both functional and non-functional 4) [16] Class Diagrams The five key processes involved in an FDD are:

1- Developing an overall model

2- Building a feature list

3- Planning by feature

4- Designing by feature

5- Building by feature

2.3.4 Adapted Agile Models

Agile methodologies are ideal for developing software with a focus on delivering working modules iteratively and frequently, but depending on the nature of the product that we are building, the requirements of the stakeholders involved, the work culture and environment in which those softwares are being built, we may need to make some changes to the agile model that's being followed. [11]

We looked at a variety of alternative adapted models, such as Simplified Extreme Programming (SXP) and Simplified Feature Driven Development (SFDD) [17], as well as other methods that took into account certain blended agile approaches based on current agile models. [11] The aim of these suggested models was to address some of the problems that have been identified in XP and FDD, such as a lack of appropriate documentation, a clear architectural plan, a proper designing phase, and confusing requirement collecting advice. However, there was insufficient data to suggest the impact of these proposed models when implemented on a variety of complex projects

of varying natures, and some of the points that remained unclear included the quality control process and how these models will perform if the software development team is not co-located and a number of team members were working remotely, sometimes even in different time zones and different countries.

2.3.5 Benefits of Agile Development

Agile software development is used by a number of organizations worldwide in developing different sorts of software projects ranging from small scale software tools to large scale software projects, with varying amounts of complexity, and different sorts of people involved in those projects and agile teams. There would be some agile team which may benefit more compared to some other teams, owing to various things like the individual experience of the people involved in a team to the overall development environment and the agile model used. Thus, for different teams and individual members the benefits of agile may be quite subjective. [35] Here, we will discuss some very key benefits that people working in an agile environment can enjoy.

2.3.5.1 Accommodating Changing Requirements

One of the major advantages of agile development is its ability to accommodate the changings requirements. The traditional software development makes it challenging to implement new requirements in the later stages of SDLC, while agile welcomes changing requirements [5]. Even if the requirements are not crystal clear at the beginning of the project, as the projects go on and both the development team get a better sense of the requirements, the agile development will help in integrating those changings in the upcoming iterations.

2.3.5.2 Customer Satisfaction

Another major advantage of agile development is that the customer is a part of the team and is actively involved throughout the project's lifecycle. This can reduce the overheads of excessive requirements engineering, risk assessment and documentation at the initial phase of the project. The customer collaborates with the members of the team frequently and this helps the development team in getting timely feedback on the implementation they've done which makes sure that the product that's developed at the closure of the project actually reflects the true needs of the client.

2.3.5.3 Qualitative Product

Software quality is another aspect where agile greatly helps the development team. As the project is developed in small and frequent iteration with each iteration having its own set of requirements and tests along with an immediate feedback from the client, the faults and defects are found earlier than that of traditional development where testing only happens at the closure phase of the project.

2.3.5.4 Incremental Delivery

The software product is delivered in an iterative manner where a complex software project is built in small, time boxed iterations. This helps the development team to identify bugs, risks and bottlenecks in both implementation and requirements, in a timely fashion and this kind of an approach is also beneficial when the client is not clear on the requirements at the beginning of the project and some kind of a working product is needed to get a better understating of the product.

2.3.5.5 Continuous Integration

Another integral part of agile methodologies is continuous integration, where new features and improvement are constantly being added to the software product being built and it helps in achieving better quality and timely integration of the client feedback [36]

2.4 Software Quality Assurance

As we have pointed out in our previous sections, developing a software product is a process that requires a set of appropriate engineering practices, which may vary based on the type and complexity of the project. The process of developing software becomes very streamlined once we have the proper software engineering practices in place and the development team is experienced enough to follow them. The chances of the project being delayed or over budget, and eventually resulting in a failure, are greatly reduced. However, [7][18] this is not the only element that guarantees our project's success. Strong Software Quality Assurance (SQA) processes are needed to ensure that the product produced at the conclusion of a time-boxed engineering process is a qualitative project that meets the requirements of the customer and the stakeholders involved.

SQA is the process of quantitatively evaluating the product being produced, and while working in an agile framework, where sprints or iterations are short, time-boxed, and repeatable entities, it's important to understand what kind of SQA procedures we'll need to ensure the product's quality.

One of the most important aspects of the SQA process is software testing, which consists of validation and verification methods, used to ensure that the software product we are creating is free of bugs, flaws and defects. [19] [20]

When we examine the different agile models, we can see that the testing process is not very well defined, and there are no clear instructions on how, when, and what types of tests should be performed at various stages in that agile model. There are models where developers are encouraged to create test cases prior to the creation of the product, and then testers and developers work together to conduct integration testing, but there are no more specifics instructions on any high-level testing [21][22]. Various publications [23][24][25][26] have pointed to some significant flaws in the XP testing methodology.

Apart from quality assurance in the product that is being developed, quality management in the process that is being followed is also required, with quality management personnel responsible for ensuring that the defined engineering practices are properly followed by the team, and the concept is to train a new team or a team member on any new modifications. [18]

Another part of the SQA process occurs when the product is pushed out into its intended environment after a sprint or a specific number of sprints, and the users for whom the project was created begin to use it, and the development team begins to get input from the end user. The significance of incorporating user input is critical, and doing so in a timely manner will guarantee that the software quality continues to improve as the sprints continue and the project is finished. This necessitates the adoption of a user-centered design (UCD) strategy. A UDC approach allocates a number of resources to studying end-user behaviors and expectations, both during the early stages of the project and after a prototype or new version is rolled out to the userbase [27].

2.5 Remote Work and Agile Software Development

In the start of 2020, the world faced an outbreak of the COVID-19 virus and it posed a great challenge for the agile teams which were using the practices that needed the team to be co-located. There was no possibility of face-to-face meetings between the members of the team and the client was not able to sit under the same roof as that of the agile development team. Various other agile practices like the concept of pair programming, where two developers work on the code for the

same functionality in a bid to timely catch errors and defects and make sure that the quality of the software is up to the mark, was no longer a possibility [4][37]. Other core agile practices like daily scrums, where members of the agile team conduct a daily face-to-face meeting at the start of each day, were not possible as well [38]

During the pandemic outbreak, it became increasingly difficult for the members of the team to collaborate face to face as the team members had to shift from working in an office to working from home (WFH). This also made it difficult from clients to collaborate with the team members as effectively as they did it while working in a co-located environment.

There were some indications that while WFH, the team members were able to grasp the requirements of the project better and the quality of the written code improved as well [39] However, this may depend upon the experience of individual agile teams, as some teams around the world were already working remotely as bigger organization and businesses found it beneficial to have a software teams working remotely as they can provide the advantage of lower development cost of the software and the ability to have experienced resources working to develop their softwares.

2.6 Summary

A brief background for this study was addressed in this chapter, as well as the level of research that has previously been done on this issue. We looked at the software engineering process and how it varied from conventional software development and agile software development. We also looked at the necessity of having strong quality assurance practices in place and examined different agile models such as Scrum, XP, and FDD, as well as the advantages they offered and the benefits of agile techniques as a whole. The purpose of this chapter is to provide the reader a clear foundation that will aid them in better understanding the different elements of the study in the following chapters.

Chapter 3

3. PROPOSED MODEL

The agile methodologies provided us with various benefits of iterative development, closed customer collaboration, fast paced development cycle and feature based development. However, there were some areas like remote working environment, user centered design and the quality control process for small-scale softwares, which were integral to our development model and the existing agile models did not provide us with any clear set of processes for those areas. To overcome those challenges and to have a defined process in place for those areas, we introduced our proposed modifications to the existing agile models which helped us in timely delivery of a qualitative product. The proposed model and the modifications made to the existing agile models, will be the primary focus of this chapter.

3.1 Motivation for a Proposed Model

A set of specified processes, which will assist us in the execution of the agreed-upon requirements for a software project, is required once the specifications for a software project are finalized. We found that traditional approaches to software development were not a good fit for our projects, which were small to medium in scale and consisted primarily of tools and utilities for the Android platform which is consistently being updated to incorporate new technologies.

Once the requirements were identified and documented, an evaluation was conducted using existing SDLCs, with the evaluation taking into consideration various aspects such as the scope of the projects, the estimated time to delivery, and the expectations of the customers. While the platform updates are generally beneficial to the developers, as they provide the developers with a chance of learning about new technologies and developing software for them, one disadvantage of the frequent and necessary updates is that the functionality and capabilities of an application may be restricted to the point where the developers must return to the drawing board and rethink the solution, if they want to remain competitive in the market for that specific tool as the new update

may include a security patch which may limit some functionalities regarding access of user's data in a way which was previously allowed. This leads us to requirements which would be continuously changing, and in order to meet the goals and keep the product seamlessly usable, we would have to return to the requirements phase for some of the features, which indicated that the general agile methods were a better and more obvious option in this instance.

As we've discussed previously, the majority of Agile models require that the team members be physically present under the same roof to ensure that the model is fully implemented in its essence. This is because various practices such as daily standup meetings in SCRUM, client collaboration and pair programming in XP will only be successful if the team members are physically present. Although we had a client who communicated with us remotely, we also had a number of people working on the project from different time zones, so it was critical for us that any model that we presented be flexible enough to be implemented in a semi or fully-remote working environment.

Choosing the Agile framework for our project had another advantage; we were able to complete all of the planning and implementation for the first release (v1) with a core set of features in less than 4 to 6 weeks, which allowed us to complete all of the project's planning and implementation in that time frame. The requirements for the whole application were finalized at the outset of the project, forming the basis of our backlog, and then the feature list for release v1 was finalized by prioritizing the requirements in that backlog. We would be pushed to complete each step of the software development life cycle in a foolproof and timely manner, with the emphasis on providing features rather than delivering the whole software in one single iteration. Agile techniques are built on iterative development and would have been a perfect match for a feature-based development strategy.

"Customer cooperation takes precedence over contract negotiation," according to the Agile manifesto. According to the [5], the customer is invited and encouraged to participate throughout the whole lifecycle. The sooner we get feedback on our implementation, the higher the likelihood of customer satisfaction, and the shorter the time it takes for customers to accept our solution after the implementation phase is complete.

3.2 Types of Software and Quality Assurance Perspective

It is essential to note that the majority of the software products on which our team has focused their efforts have been small size mobile applications for the Android operating system. These products are prompted by various customers and consist mostly of tools and utilities such as a data recovery application, a virtual private network (VPN), or an application that assists families in tracking down individual members via the use of Google Maps and backend rest services.

Along with the timely release of the software, another important requirement for us was the development of qualitative applications using the UCD approach. As you may be aware, in most small-scale software products, particularly in the mobile applications arena, the customer who initiated the development of the software is not always the same customer who is actually using the software product. One such instance would be the development of a data recovery application for the Android mobile platform, which is brought to us by a business that wants to publish the application in the mobile app store and, as a result, make some money via the use of advertising networks. The product that we create would be accessible to every user in the world who has an Android phone, and therefore the main users of that app would be those who download it and use it to recover their lost data.

To model the behaviors and usage patterns of the end-user in such an approach, prior to development of the application, would be a difficult aspect because there are hundreds of Android operating system vendors, like Samsung, OnePlus, Huawei etc. [40], each with their own implementation of the opensource Android platform, and ensuring that our software works seamlessly on all of those different vendors would be a significant challenge. It was discovered during our research of various agile models that little or no knowledge existed on how to properly address the issue of testing and how to properly implement a UCD approach in situations where it is not possible to completely model the end user and the intended operating system environment before developing software. Using an agile framework, we will explain how the implementation of an SQA process, as well as the actors and stakeholders that are engaged, was done, in an effort to simplify the SQA process even more effectively.

Now that we have a better grasp of the difficulties and incentives associated with adopting our suggested changes to the current agile models, we will proceed to discuss the specifics of our proposed model in more detail.

3.3 Agile Team and Key Players

In any agile model that we examine, there are always a few core roles and responsibilities that are clearly defined and can be used to get a better understanding of the types and numbers of resources that are needed in a successful agile team. Depending on which agile model is being used, the roles and terminology used to define these specific roles may differ. For example, in Scrum, there is no concept of a traditional project manager; instead, the role of product owner is assumed by the Scrum team, whose primary responsibility is to keep the development team on track and ensure that the timelines that were agreed upon before the start of the project or a sprint are respected by the entire team. Yet another example would be the role of the client; in Scrum, the role of the client is taken on by the Scrum Master, who is responsible for such things as bringing the viewpoint of the client to light as well as having the power to prioritize specific items in the product backlog. [28] Table 1 provides some insight into the fundamental distinctions between jobs and responsibilities across the different models under consideration in this article.

Table 1 - Jobs and Duties

Jobs and Duties			
SCRUM	**FDD**	**XP**	**Adapted**
In the SCRUM model, some roles are specified, such as SCRUM Master, Product Owner, and other team members. [29]	In FDD, there are also specified roles that are split into essential roles, supporting roles, and extra roles. [30]	In XP model, the code ownership is seen as collective, with no specified responsibilities. [29]	We modified the SCUM and FDD models to include certain specified responsibilities in our proposed model, which will be addressed in the pages that follow

Some models, such as XP, do not have predefined roles, with the goal of ensuring that the team, rather than an individual or a group of individuals, owns the project as a whole. However, because our projects are being worked on in an environment where each team follows a well-structured hierarchy, we need to have defined roles and responsibilities so that everyone knows what job they are responsible for. We have established teams in our proposed model, and each team includes additional roles/people that are a member of that team. In our concept, there are three major groups:

1- Core Team
2- Management Team
3- Supporting Team

The three major teams that are included in our proposed model, as well as the responsibilities associated with each team, are shown in Table 2.

Table 2 - Agile Team

Roles definition		
Core Team	**Management Team**	**Supporting Team**
Chief Programmer	Project Manager	Supporting Programmers
Chief Architect	Client Representative	Supporting Designers
Chief Designer		Supporting Testers
Chief Tester		IT Support Engineer

These roles are established in light of the nature of small-scale projects, and the amount of resources assigned to each of these jobs varies based on the project, the anticipated timeframes, and the project complexity. We can have a different number of resources for each role, but some roles will be bound to a specific number for the sake of clarity and a structured hierarchy, such as the role of a chief programmer; one project can only have one chief programmer at a time, as that person will be leading the team of developers working on the project. Table 3 defines the breakdown of the amount of resources required for each role and their main duties.

Table 3 - Roles and Responsibilities

Role	Resources	Key Responsibilities
Chief Programmer	1	The chief programmer is in charge of the project's coding and development efforts, as well as directing the development team's work and being actively engaged in all phases of the project.
Chief Architect	1	It is the chief architect's responsibility to lead the design team through all stages of the project.
Chief Designer	1	This position is in charge of directing the design team, which is in charge of designing the project's User Interface (UI), and is mostly engaged in the sprint's development phase.
Chief Tester	1	This position is in charge of developing test cases and, subsequently, testing the project's implementation. This role plays a part in all stages of development.
Project Manager	1 - 2	This position is in charge of keeping the development team on track with the sprint objectives. This position, like the sprint master, is in charge of discussing and maintaining daily objectives for the development team through face-to-face meetings and sometimes an online daily scrum form [13]
Client Representative	1 - 2	The client representative represents the client's or end-user's viewpoint to the team. This position is engaged in both initial and sprint planning phase and may propose backlog prioritization. The project manager is typically the one to contact.
Supporting Programmers	2-3	These supporting programmers work under the direction of the chief programmer to code the software being created and are actively engaged in the project's development phase.

Supporting Designer	2-3	These supporting designers work with the main designer to build the user interface for the software under development and are actively engaged in the project's development process.
Supporting Testers	1-3	These supporting testers are headed by the chief tester and are responsible for ensuring the product's quality by creating test cases early on and then testing features as they are created.
IT Support Engineer	1	This role would be responsible for providing IT support to the whole team. This can be maintaining and fixing any issues that may arise with a personal computer or laptop which is being used by the team or any software or OS related issues.

3.4 Initial Requirements

Over the course of our projects, and after studying various agile models such as Scrum, XP, and FDD, we began to develop our own sense of how these models worked, what their advantages were, and what areas the models lacked in providing us with the assistance that we required, so we began to work on making some modifications to the existing models and eventually began defining our own. In our suggested approach, we prioritize delivering functional versions of our projects in every scheduled release, allowing us to develop a number of hand-picked features in a single sprint. Rather than having a list of requirements to implement, the emphasis is on features, thus a feature backlog is established. All of the requirements are thus organized into features, which are then chosen for release implementation. Table 4 shows the fundamental differences in initial requirements across the various agile models under study.

Table 4 - Initial Requirements

SCRUM	**FDD**	**XP**	**Adapted**
All the requirements that must be	All of the features are organized into	An overall release strategy gets	All of the project's needs are organized

included in our application, are documented.	sets according to subject areas.	established, as well as the overall scope of the project.	into features, which are tracked in a feature backlog.

3.5 Phases in the Proposed Model

The model we propose is highly iterative in nature and there are three major phases that happen in the course of a project, which are stated as follows:

1- Initiation
2- Execution
3- Retrospection

We will look at each of these phases in detail, along with the actors and the steps involved in each of these phases.

3.5.1 Initiation

This is the first phase of our suggested model, which starts after the development team has formally begun the project and we have a basic understanding of the product that the customer requires. This is the phase in which the emphasis is on establishing an overarching vision for the product that we will create, and the project requirements may not be very clear to both the customer and the development team, at this point in the project. Here, a process for collecting requirements, understanding the project as a whole, and knowing the end user and the environment in which our software will be deployed is begun with the assistance of the product manager and the client representative. Table 3 delves into these specific roles and the teams that are engaged in our agile framework.

Another important aspect of this model is user-centered design (UCD), and during this phase, before the features and requirements list are created, the core team and management team investigate similar software which are currently available in the market so that we can get a good idea of what the users of this domain like to have included in such a product. All findings are documented in an online document that is shared with the entire team, and meetings are held to discuss any unique and useful features. This practice aids the development team in developing a

better understanding of the concept of the product they are developing, as well as writing some of the functional and non-functional requirements.

The product manager's role in the Scrum model is to begin compiling a list of both functional and non-functional needs and establishing a product's vision by holding meetings with customers and other project stakeholders. In our suggested approach, the management team would be responsible for getting the requirements started and having a comprehensive conversation with the stakeholders to have the high-level requirements written down and documented online where the members of the team can easily access those requirements.

Once we have a set of high-level requirements in place, the management team and core team conduct a series of meetings, which may be face-to-face or online, where the requirements are divided into features. For the sake of an example, let's assume that the Android platform requires a "Data Recovery" application that can help users to recover some of the lost data which was accidentally deleted. A few of the requirements for such a product can be as follows:

- R1: A user may scan a particular directory using the scanning function.
- R2: The user may apply a filter function to look for files in a particular date range under a specific directory.
- R3: During the scanning process, the User Interface (UI) should not get stuck.

R1 and R2 are functional requirements, while R3 is a non-functional requirement. All of these requirements, on the other hand, relate to a particular feature of "Directory Scan" and may be readily grouped together. The chief architect, chief designer, and chief programmer suggest grouping requirements into features, while the client representative and project manager are key players in verifying each feature and its requirements, and they have the authority to add or remove any number of requirements from a feature.

3.5.1.1 GitHub Issue Tickets

We use GitHub to manage our list of features and their needs, which is one of the main components of our model. To ensure that the requirements are accessible to every actor engaged in this phase, whether remote or co-located, we use it to manage our list of features and their requirements. A GitHub issue ticket is generated for each feature, and each issue ticket we produce during this phase will relate to one of our product's features. Every issue ticket includes a unique ticket

number, a title, a list of requirements, user stories connected with that feature, acceptance criteria, and the tests that feature requires, as well as an estimate of how long it will take to build that feature. Table 5 provides a comprehensive description of the various features of our GitHub issue ticket.

Table 5 - Features of GitHub Ticket

Ticket Features	**Details**
Ticket Number	This is the unique ticket number assigned to each GitHub issue, and it helps us in keeping track of each feature.
Issue Title	This will be the feature's name, such as "Directory Scan" in our earlier example.
Description	Every GitHub issue includes a description, and in our case, here is where we'll describe all of the feature's functional and non-functional needs.
User Stories Comment	The management team and the core team work together to develop the user stories. On the GitHub issue, the project manager adds a comment which may have a link to an online user stories document.
Acceptance Criteria Comment	The chief architect, chief tester, client representative, and project manager have a series of meetings to establish the feature's acceptance criteria, which are then placed as a comment on the issue ticket.
Test Cases Comment	The Chief tester and, on occasion, a supporting tester write out the test cases connected with that feature. These are the cases that will be validated after the feature has been implemented in the execution phase, and they do not have to be low-level test cases like unit tests.
Estimated Time	Members of the core team collaborate and tag each feature with the anticipated number of days it will take to develop and test the feature.

We'll have a feature backlog in place after we've specified a set of features and filed issue tickets on GitHub for each one. The following step of the proposed model, "execution," will use these features as input. This phase is only considered complete when all of the product's necessary features have been documented and shared with the team.

3.5.2 Execution

This is the second phase of our proposed model, and it is during this phase that we begin our real development effort. This phase comprise of a number of sprints that we may need to complete before we can execute our feature backlog for a final release.

The sprint is a time-boxed development effort, and the amount of time allotted to a sprint varies across projects. In the case of our products, each sprint lasted 2 to 3 weeks, during which time we implemented the prioritized feature backlog and released a functional version of the product after every sprint. Each sprint is further split into three sub-phases; Sprint Planning, Development Phase, Sprint Closure.

3.5.2.1 Sprint Planning

The aim of this initial sub-phase of the execution phase is to plan for the development phase. We now have a feature backlog for our product, and members of the management team, such as the client representative and project manager, are in charge of prioritizing the features. Each feature is tagged with the estimated time it will take to implement and test that feature, and we also know that the sprint will be a time-boxed effort, so the management team handpicks a set of features to be implemented in that sprint, keeping in mind the duration of the sprint and the estimated time for each feature. The features chosen are prioritized because the client may want certain particular features to be pushed out ahead of the others so that we may gather user feedback and prepare any necessary improvements. We use the GitHub Kanban boards [41] to organize the issue tickets which are created. A normal Kanban board will consist of four columns; To-Do Tasks, In Progress, Ready for QA and Finished Tasks. The features that are hand-picked during the sprint planning phase are transferred to the GitHub Kanban board's "To-do" column. The sprint plan cannot be modified after the prioritized feature backlog is in place and the team knows all of the features that will be deployed in the sprint. The sprint planning step takes around 1 to 2 days to complete. With

the sprint plan in place, we can move on to the second sub-phase phase of our model's execution phase which is the development phase.

3.5.2.1.1 User Centered Design (UDC)

We utilize analytics services that give us a lot of useful data for the UDC approach and analyzing user habits for our products. We track user input for those features once some of the backlog items are implemented in a release and the project is pushed out to users. In our project, we've incorporated Google's Firebase analytics services [42], which may help us figure out which features are the most popular with users. If we consider a mobile application with several screens displaying different types of data, one screen might be a "Profile Page" displaying user information, while another could be a "Friends List" displaying a list of the signed in user's friends. Now, if the option to view the friends list screen from the profile page is available, firebase analytics shows us the pattern of how many people navigated to the profile page and what percentage of those users clicked on the "Friends list" option. This allows us to identify our app's usage patterns and most popular features, allowing us to better understand our users.

These analytics may also tell us whether our software has crashed as a result of a code error, as well as the details of the mobile device on which the crash or error occurred, number of users impacted, and Android version on which the incident happened. These analytics are the foundation of a user-centered design, and as soon as the first version is made available, we begin to see trends and begin to offer value to consumers by correcting any problems that occur often and enhancing features that are popular among the product's users.

3.5.2.2 Development Phase

This is the part of a sprint when the real development work begins and programmers, designers and testers actively collaborate on the creation of the prioritized features list that was agreed on during the sprint planning phase. This is the phase where the majority of the sprint's allotted time will be spent, and it will last around 2 to 2.5 weeks. We've taken some cues from the XP method, which sees designers, testers, and programmers collaborate on feature development. The designers continue to create the user interface designs required for the backlog features, the developers begin developing code for those features, and the testers begin preparing test cases to be run after the features have been implemented and are ready for QA.

Once the development phase has begun, development team members must write daily stand-ups at the start of each day, using the Scrum standup method as inspiration. The difference is that Scrum requires us to conduct these standups as a 10- to 15-minute, face-to-face, meeting at the start of each day, while our standups are written and each member of the development team is required to write their daily standups in a particular manner. A standup message is made up of three parts, which are mentioned below:

- Done Yesterday: The developers write the job that they completed the day before.
- Today's Tasks: The tasks for today are mentioned below.
- Blockers: Any issues that developers may encounter are mentioned here; these issues may range from a poor internet connection to a blocker triggered by a server being down.

The XP approach employs pair programming, in which two developers collaborate on a single function's code, with one developer writing the code and the other reviewing the code produced by the first developer to minimize the likelihood of errors and mistakes, according to [31]. We take inspiration from the agile model XP and use GitHub pull request (PR) to implement code review. A developer develops the code for a particular function and then submits it to GitHub as a pull request and requests a review by another developer on the team. This way, any code written during the development phase is reviewed by at least one other developer. Another benefit of this approach is that the two developers, i.e. the coder and the reviewer, do not have to be sitting together under the same roof and can review the work in a seamless and efficient manner. Any feedback from the reviewer is placed on the pull request and is publicly visible to all members of the team, and the reviewer may even point out the exact line of code containing an issue.

There may be many pull requests for a single feature since there may be multiple functional needs connected with it. It is best practice to make pull requests to be as brief as possible for ease of review, and a common practice is to have a pull request for each functional requirement. When a reviewer approves the pull request, the code in that pull request is merged and integrated with the rest of the existing codebase, and once all of the features' requirements have been developed and reviewed, the GitHub ticket for that feature is moved to the "Ready for QA" column in the Kanban board, signaling to the tester that the feature can now be tested using the test cases written for that feature.

As soon as the feature is marked as 'Ready for QA' on the Kanban board, the testing team begins running the test cases they've created for each of the feature's requirements. Table 6 contains an example of a test case.

Table 6 - Sample Test Case

Requirement Type	Functional
Requirement	The user should be able to input up to 20 small case alphanumeric characters in the username field.
Step(s) for testing	1- The user is limited to 25 alphanumeric characters in this field. 2- The user may input precisely 25 alphanumeric characters in this area. 3- If the user attempts to enter a 26th alphanumeric character, the field limits the input.
Status	Status can be either Passes or Failing
Comments	The testers record any comments they may have regarding the test case here.

If any issues are discovered during the implementation of any features, the testers communicate them to the development team, the feature ticket is moved back to the "To-Do" column, and the test case feedback is commented on that issue ticket, indicating to the development team that they need to work on fixing these issues, and the ticket is moved back to the "Ready for QA" column once the issues are resolved. The ticket is moved to the "Finished Tasks" column and declared as closed after all of the test cases have been passed.

This procedure is repeated for all of the features in the "To-do" column for that sprint, and the tester is continuously testing each requirement and feature, as it is implemented to ensure that nothing breaks. If there are any roadblocks or technical difficulties encountered during the development of a feature, they are recorded as a remark on the feature's issue ticket. A non-functional Application Programming Interface (API) that was critical for that feature is an example of a blocker. For example, we were developing an app that showed the user the current weather of the day and we were planning to use a specific data API directed by the client, but once the

implementation work began, the API became non-functional and thus could not be used. This may be a roadblock for implementing the weather feature; it will be noted under that feature, and we may need to either request another API or put the work on this problem on hold until a later sprint.

When the prioritized list of features that was decided during the sprint planning phase is developed and tested, and the code for the related requirements is merged into a single codebase, a beta release including all of those features is produced, and the development phase is rendered as complete.

3.5.2.3 Sprint Closure

This is the sprint's last phase, and the management team and the client representative are the main actors. The beta release that was shared with the team is currently being tested for all of the new features, as well as all of the acceptance criteria connected with each feature. The lessons learned throughout the development phase, as well as any roadblocks encountered, are recorded in an online document. All team members have access to this document, and they are invited to give their unique contributions at this time. The sprint's findings are used to redefine the feature backlog. If the acceptance requirements are met and the sprint is considered successful, an alpha release is produced; otherwise, all findings and required improvements or features are added to the product backlog and no alpha release is made, indicating a failed sprint. In several of our projects, when speed to market was important, an alpha release was produced after each sprint, and if an issue with a particular feature arose, just that item was removed from the created alpha release, and the necessary improvements were put to the queue for execution in the following sprint.

3.5.3 Retrospection

This is the closure phase of our proposed model and once. After all sprints have been completed and all features from the product backlog have been implemented, the project's final release is rolled out and tested against all acceptance criteria. The release created during this phase should ideally contain all of the project's intended features, implemented and tested. The project will be deemed complete if the acceptance criteria established in the Initiation phase were met, and a final rollout will commence in the project's intended environment. After that, only maintenance work will be performed on the project. We also record the project's lessons learned, which are shared with the whole team in the form of a shared web document. The user documentation, training materials, and any marketing materials may be created here, and once this phase is completed, the

project is completed and moved onto the maintenance phase, which is outside the scope of the current research, which is primarily focused on the development of software from conception to deployment.

3.6 Key Difference Between Models Under Study

All agile models may have various practices, as well as different kinds of projects that they are best suited for, different team sizes, and different practices that each model follows. Our agile model is based on existing models such as Scrum, FDD, and XP; there were several key features in those models that were chosen as they were defined in those models, some features were discarded because they did not fit our needs, and some features were modified to fit the nature of our projects. Table 7 details some of the changes and adaptations that have been made to existing agile models and it also gives a chance to have a look at how those practices differed between each existing model as well.

Table 7 - Key Differences between Agile Models

Scrum	**FDD**	**XP**	**Adapted Model**
A product backlog is a list of the requirements that must be met in order for a project to be completed. [29]	Features are organized into sets and topic areas, and planning is done by feature.	The project's general development strategy is created, and the releases are planned. [31]	Before the project begins, the requirements are organized into features and a feature backlog is established.
The emphasis is on the incremental creation of a larger system.	The emphasis is on development based on prioritized set of features.	Iterative and incremental development [13] The project is constantly being developed and constructed, with	A functioning product is created when a sprint is completed and the planned features are implemented. The product is delivered to the end user after a sprint is completed. In a single

		an emphasis on minor releases.	sprint, the desired features are created and tested.
SCRUM does not define testing. [13] Each sprint item goes through a testing cycle.	After the development process, the whole feature is tested.	Unit and acceptance testing are performed on a regular basis. [13][31]	The testing is meant for the whole feature but is done more often and many times in a single sprint, drawing inspiration from both FDD and XP. The developer and the tester collaborate closely to eliminate code mistakes, after which the project manager, tester, and client representative conduct acceptance testing.
Sprints last around 4 weeks.	An iteration lasts between one and three weeks.	An iteration lasts between one and three weeks.	A sprint will last two to three weeks.
The number of team members ranges from two to ten.	Team Size: is undefined	The number of team members ranges from 10 and above [31]	The number of team members ranges from 10 to 15.
Members of the team should be based in the same location.	Members of the team should be based in the same location.	Members of the team should be based in the same location.	Members of the team may work in the same location or on remote basis.
It is not essential to pair program.	It is not essential to pair program.	The XP model's main characteristic	We presented the idea of pair programming in

		is pair programming.	terms of Pull Request (PR) reviews, based on the XP model, in which one programmer implements a piece of code and then writes a GitHub pull request for it, which the other programmers evaluate and test before accepting the PR.
Changes during iteration are not permitted.	Changes during iteration are not permitted.	It is permissible to change throughout iteration.	Changes are not allowed during iteration, but they must be recorded and applied in the following sprint.
The SCRUM concept is built upon daily scrums. The daily meeting is a 10- to 15-minute face-to-face meeting before the start of the day.	Instead of regular conversation, FDD relies on documentation.	Face-to-face standup meetings are held on a daily basis.	Daily stand-ups are conducted in a textual style in order to include workers working remotely. Tasks accomplished yesterday, tasks scheduled for today, and blockers are all included in the written stand ups.
In SCRUM, there is less documentation and more emphasis on	Within the team, documentation is the main means of communication.	There is little documentation, and the emphasis is on development and	Although documentation is not the main emphasis, each work completed and planned is recorded throughout the project in

face-to-face meetings.		daily stand-ups for communication.	the form of written standups, sprint reviews, and feature backlog.
Each sprint has its own set of acceptance criteria.	There is an acceptance criterion for each feature.	There is an acceptance criterion for each iteration.	The release acceptance requirements are established during sprint planning, while the overall acceptance criteria are defined during the preparation phase.
The SCRUM team determines the development order.	The modelling authority and domain specialists set the development sequence. [32]	The client specifies the development sequence.	Following a series of meetings, the customer, project manager, and core team determine the development order jointly.
It may be utilized for any size project.	Typically used for medium to large-scale projects.	Appropriate for small-scale projects	Suitable for small to medium-sized projects with fluctuating requirements.

3.7 Quality Assurance Practices

Testing of the software to check and validate that there are no defective or erroneous states in the app is one of the important techniques employed in any agile model to guarantee the quality of the product being created. In most agile models, there is unit testing, acceptance testing, and integration testing. We get a notion of test-driven development in XP where the test cases are created prior to the code itself and when a piece of code is produced, it is also regarded as legitimate when those test cases pass. Quality isn't simply determined by testing; additional techniques such as collaborative code ownership, pair programming, onsite-customer, and code reviews ensure that

the software being developed meets the required standards. We've gone through a number of the techniques that we employ in our model to guarantee software quality, such as the idea of programmers evaluating each other's code and a UCD that uses real-time usage statistics to improve software quality. We shall compare the different quality assurance methods available in the agile models under investigation in Table 8

Table 8 - Quality Assurance Practices Among Models

Quality Assurance Practice	**Scrum Model**	**FDD Model**	**XP Model**	**Proposed Model**
Customer as a member of the team	✓	✓	✓	✓
Joint Meeting for Planning		✓		✓
Inspections of the Code		✓		✓
User Centered Design Approach				✓
PR Reviews				✓
Pre-defined Test Cases			✓	✓
Code Ownership being Collective			✓	✓
Real-time usage statistics				✓
Unit Testing	✓	✓	✓	✓
Acceptance Testing	✓	✓	✓	✓
Integration Testing	✓	✓	✓	✓

3.8 Summary

In this chapter, we went through the various modifications that were made to the current agile models, in dept. We delved into the nitty gritty details of our proposed model, including how the requirements are structured, who the main actors are in our proposed model, and the different stages and related procedures that are needed for the creation of the software product from initiation through development. We also highlighted how the user-centered design approach was used to provide quality control and what extra advantages our model offered in terms of software quality assurance procedures.

Chapter 4

4. Remote Agile – Challenges and Proposed Solutions

4.1 Chapter Outline

This section of the research will only be focused on the teams who had no experience of remote work before the COVID-19 outbreak but were experienced with the agile way of developing the software. This chapter will look at what challenges the developers around the globe faced while a sudden shift in work modality forced the members of their team to work in a WFH environment. Then, as we discuss the challenges faced by the software developers, we will propose some possible solutions to those challenges that we implemented in our agile team and then there is going to be some analysis on how those proposed solutions helped us in overcoming those challenges.

4.2 Challenges and Proposed Solutions

An agile team working with a defined agile model in an office environment, may face a number of challenges if the work environment suddenly changes from an office to remote work. There can be a number of factors which may contribute to these challenges, like the complexity of the product being developed, the experience that the team has in working with different agile models and a lack of proper remote working practices in the agile model that's being followed by the team as most of the models like Scrum or XP do not cater that well for remote working environment and to get good results from those models, a co-located team is a necessity. Here, we will discuss some of the challenges that an agile team may face, when working with an agile model which requires the team to be co-located and suggest some proposed solutions for overcoming those challenges.

4.2.1 Reporting and Progress Meetings

Most of the agile models like Scrum rely very heavily on face to face and in-person communication. If we take a look at Scrum model alone, there are daily scrum meetings which are

supposed to happen at the start of every working day where all members of the team have a face to face meeting to discuss progress, plan for the day and discuss any pending blockers [43] This meeting is also called "Daily Standup Meeting" and is usually 15 minutes in length, such a meeting is not possible for the teams which are working remotely. An obvious solution to cater for this problem is to get an online meeting going where team member join an online audio or video call and have their daily stand-up but in-case the members of the team are located in different time zones around the globe, such a meeting can very easily become unstructured and lose its significance. The proposed solution to overcoming this problem is to have a written standup which is accessible to all members of the team, coupled with an online video conference at the time of the day which is most suitable for all members of the team. Table 9 summarizes the challenges and the proposed solutions for the reporting and progress meeting practices in a remote environment.

Table 9 - Reporting and Progress Meeting Challenges and Proposed Solution

Agile Practice	**Goals**	**Challenges**	**Proposed Solution**
Daily Standup and Reporting	• A daily face-to-face progress meeting • Discussing what was done previously • Discussing any pending blockers • Discussion of what is planned for the day ahead	• Face-to-face meetings not possible for teams working remotely. • Team members can be in different time zones and have different working hours. • Meetings can become unstructured conversations	• Daily written standup where every member of the team lists down "tasks done previously", "Blocker faced (if any)" and "tasks planned for today" before they start their day. • An online video conference where all members of the team have a call and the progress is discussed.

			• Video conferences can be done at the time suitable for every team member and the time is increased from 15 minutes to 30 minutes for a detailed conversation and to have room for discussing any pending blockers.

4.2.2 Documentation

Agile methodologies focus on reducing the overheads associated with traditional software development and one of those overheads is excessive documentation. However, in every software project there are a number of documents which are essential to the development efforts, like the use-case model and test cases that the developer needs to keep in mind while developing the application. In a co-located environment, the sharing of those documents within the team in form of handouts, and meeting notes can be done very efficiently and aligning the team with the project's vision is also less technical as compared to a remote working environment. To overcome this issue, a very clear approach needs to be taken where every member of the team has access to the needed documents and they are aligned with the goals set for a sprint or the overall vision of the project. In this case, the proposed solution was using an online project tracking system like GitHub where we create a GitHub board [41] to manage the overall requirements backlog and each individual requirement, by creating separate GitHub issues for each action item and each functional and non-functional requirement. All team members have access to those Issues and can comment on any ambiguities or ask questions under each issue, and also have an option to tag the individual that can help them better understand that requirement or action item. Table 10 summarizes the challenges and the proposed solutions for the documentation practice in a remote environment.

Table 10 - Documentation Challenges and Proposed Solutions

Agile Practice	Goals	Challenges	Proposed Solution
Documentation	• Aligning the agile team with the project's vision • Creation of necessary documents like use-cases and functional/non-functional requirements.	• Difficult to efficiently share documentation on paper, when the team is working remotely. • Challenging to address ambiguities and questions regarding documentation while working remotely	• The use of an online system like GitHub or Jira to track each individual requirement is an action item, as a separate issue. • All team members have access to a centralized source of information, where they can ask questions to other team members in case of any ambiguities.

4.2.3 Sprint Planning

At the core of agile methodologies are the time-boxed iteration, also known as sprints. Before the start of each new sprint, there is a sprint planning phase where all members of the development team, along with the client or a client representative plan ahead for the upcoming sprint. Here, the backlog is refined and the prioritized requirements or features are handpicked by the client and the team to be developed in the next sprint. There may have been some new findings in the previously concluded sprint and it may have led to some new requirements or enhancements to the current requirements as the development team and client gained knowledge and developed a better understanding of the requirements. Sprint planning requires meetings where all team members are

involved in a thorough discussion where the action items are discussed and agreed on. In a remote environment, where members can be located in different time zones, conducting such thorough conversations becomes more challenging and agreeing on action items also becomes unstructured. To overcome this problem, the suggestion is to have multiple meetings instead of a single sprint planning meeting where the members of the team are given sufficient time to prepare for the meeting beforehand and we have one meeting to discuss the action points and another for agreeing on those action points.

Table 11 - Sprint Planning Challenges and Proposed Solution

Agile Practice	**Goals**	**Challenges**	**Proposed Solution**
Sprint Planning	• Refining backlog and prioritizing the requirements to be developed in the next iteration	• Remotely conducting such a thorough meeting in one-go can become unstructured, there can be times there can be communication barriers where the team members cannot efficiently voice their inputs or concerns.	• Multiple meetings instead of one thorough sprint planning meeting. • Members of the team should be given sufficient time to prepare. • One meeting to discuss action points while another to agree on those action points.

4.2.4 Pair Programming

Pair programming is a technique where two developers work together in implementing a single requirement and such a technique can help quickly identify bugs and code defects as the requirement is being developed. This technique is not feasible when all the development team members are working remotely. To cater this issue, the concept of GitHub pull requests [44] were used where each requirement is developed by a single developer but for that coding

implementation to be merged into the existing code, another developer has to review and approve it, so that every piece of code has more than one pair of eyes to verify it to be free of bugs and defects.

Table 12 - Pair Programming Challenges and Proposed Solution

Agile Practice	**Goals**	**Challenges**	**Proposed Solution**
Pair Programming	• Two programmers working together to easily identify bugs and defects in code	• Not feasible in a remote environment as the members are not co-located.	• The use of GitHub pull-requests where one developer implements the requirements and another verifies and approves it.

4.2.5 Agile Team Coaching

An agile team which is experienced in working on an agile model in a co-located environment cannot be expected to convert to a remote working agile model overnight, as there are various practices which are going to be novel for them. A set of proper training sessions is required for the agile team where they are coached on the changes being introduced, so if they have any ambiguities or concerns, they are able to easily address those. The agile methodologies focus on team work and close collaboration and if we talk about models like Scrum and XP, most of the practices work where team members are working correctly. If there has to be a change in the work modality and we need to make modifications, it should be done in such a way where old practices are not simply discarded and replaced, but they are modified to fit the new work modality. One of the examples would be of the pair programming practice where the agile practice of pair programming is replaced by code reviews using a version management system like GitHub.

4.2.6 Centralized Information

Another very common challenge that an agile team may face while shifting from an in-house work modality to a remote work environment is that of how information regarding the project is shared with them. When the team is co-located, the collaboration between member of the team is generally very strong and thus the information sharing is also done more easily and if a member of the team needs a certain bit of information, they may go directly to the concerned role and get that information. While working remotely, it becomes much more important that the members of the team have easy access to the information. To ensure easy access, we have to make the information centralized where all members of the team can easily access it and if there has to be any changings, they are also visible to the team members. This can be done by using online documents where version history is maintained automatically and the changes are reflected to all team members who want to view the current or any previous version of that document. When it comes to requirements, we already discussed the use of issue tickets using GitHub or Jira where all the developers can view the associated documents with that requirements and can also comment and ask questions under each issue ticket.

4.2.7 Team Engagement

Another important aspect to remote working is that the team can lose motivation and get fatigued if there are multiple and long online meetings scheduled during the day where one participant is presenting while the rest have to sit through the presentation, taking notes. It's very important that the scheduled meetings are engaging, where it is encouraged that all members of the team have to contribute and give their inputs. Any meeting which is longer than an hour can have 10 to 15 minutes breaks in-between for the team members to relax, freshen up or compile any meeting notes or questions without any worries of missing out on important information.

4.3 Summary

This chapter focuses on how the agile software development process may function in a remote working environment, which was unavoidable after the COVID-19 outbreak. This study focuses on teams who have adequate expertise with agile software development methods in a co-located setting but may need to adapt their existing practices to suit the new remote context. We addressed the many difficulties that a remote working environment may provide to an agile team, as well as some potential solutions to those challenges.

Chapter 5

5. Results and Analysis

5.1 Chapter Outline

This chapter will examine the outcomes of the proposed model, which was discussed, in-depth, in Chapter 3, once it was implemented in a real-world situation for the development of new small-scale software products, and we will evaluate the outcomes of the suggested model implementation. We'll also look at how the different recommended solutions for catering to a remote working environment, described in Chapter 4, aided the agile team in their software development process.

5.2 Introduction

Our team works on a range of different projects, and we selected agile and then Scrum as our development approach. There was a lot of documentation and support material available to us [33] to enable us to smoothly integrate Scrum. Over time, our team became more adept at agile software development and identified obstacles in our Scrum approach. For example, Scrum was intended to develop a bigger system in iterations or sprints, but in our case, each iteration had to provide value in the shape of a working product. We also observed a lack of comprehensive code reviews and rigorous testing throughout the sprint, which might guarantee that the final result was free of major defects and issues.

We started studying and investigating alternative agile methods like XP and FDD to fill in the holes that Scrum left us. Such as pair programming, code review, communal code ownership, frequent releases, and feature driven development (FDD), where needs are divided into features and then produced in sprints.

We realized that a single agile model like Scrum may not be enough to apply in a real-world situation, and that additional models like XP and FDD have appealing characteristics that Scrum

lacks, and can help us solve some of the issues that emerge if we just use Scrum. Even with a blended agile approach, there were still gaps like such agile models didn't provide any guidance on how to manage the team members who were working remotely with and due to practices like daily face to face standup meetings and customer being part of the co-located team. As a result, we proposed our own agile paradigm, which we described in detail in Chapter 3.

5.3 Analysis of our Proposed Model

Here, we will provide our findings and outcomes of our proposed agile model. We have written down our project details in table 13 in terms of the anticipated and actual timeframes of a particular release of that project, as well as the proportion of practices utilized from each agile model under consideration. The percentage used to determine the utilization of a particular model is based on the number of key practices utilized from each model, and the percentage in the proposed model column are based on the practices which were either changed and adapted in some way or another from an existing agile model. The main development and quality assurance procedures for the various models are listed in tables 7 and 8. For clarity, the percentages have been floored to the closest multiple of five.

Table 13 - Projects Details and Percentage of Agile Models Used

Project Name	**Planned Timeline**	**Actual Timeline**	**Scrum (%)**	**FDD (%)**	**XP (%)**	**Proposed (%)**
Video Recovery (v1)	4 weeks	9 weeks	100%	0%	0%	0%
Video Recovery (v2)	3.5 weeks	6 weeks	80%	5%	10%	5%
Video Recovery (v3)	3 weeks	5 weeks	35%	0%	20%	45%
Video Recovery (v4)	4 weeks	5 weeks	20%	0%	20%	60%
Photo Recovery (v1)	3 weeks	5 weeks	90%	0%	10%	0%
Photo Recovery (v2)	4 weeks	8 weeks	70%	5%	5%	15%
Photo Recovery (v3)	4.5 weeks	5 weeks	25%	0%	20%	55%

Family Locator (v1)	2 weeks	6.5 weeks	90%	0%	5%	5%
Family Locator (v2)	3.5 weeks	7.5 weeks	45%	5%	5%	45%
Family Locator (v3)	4 weeks	7 weeks	15%	0%	5%	80%
Weather App (v1)	4.5 weeks	8 weeks	80%	5%	5%	10%
Weather App (v2)	5 weeks	8.5 weeks	70%	5%	15%	10%
Weather App (v3)	4 weeks	5 weeks	30%	0%	20%	50%
Backup and Restore (v1)	7 weeks	11 weeks	65%	5%	10%	20%
Backup and Restore (v2)	3 weeks	4.5 weeks	60%	5%	5%	30%
Backup and Restore (v3)	4 weeks	6.5 weeks	60%	5%	10%	25%
Backup and Restore (v4)	3 weeks	3 weeks	5%	0%	5%	90%

5.3.1 Positive Results

We used SCRUM when we first started building our projects, and if you look at the project "Video Recovery," Scrum and its practices were used 100 percent for the purpose of developing and rolling out that project during the v1 release, and despite the fact that this was the first small-scale project developed using the Scrum model, the team was still sufficiently experienced with Scrum practices.

As we go through our projects, we can see that we are beginning to integrate techniques from other models, such as XP and FDD, as well as our own changes to that model. In “Photo Recovery (v1),” for example, we integrated the idea of XP's pair programming and frequent releases, and we can see that the use of Scrum has been reduced to around 90%. We later included our changes of pair programming being replaced by GitHub pull request reviews in our v1 release of the “Family locator” project. It's easy to see how, as our projects progressed and we continued to modify

existing models, our proposed model began to take shape, and in our most recent project, "Backup and Restore (v4)," we used 90% of the development and quality assurance practices from our proposed model, with only 5% of pure Scrum and XP practices. The user stories were created in the same way that we do in SCRUM, and we continued to utilize the XP frequent release model, both of which are critical to the success of our projects.

If we take a close look at the timeline of the various projects listed in Table 13, we can see for all of the projects, except for the "Family locator" project, our proposed model worked really well and produced the results which led the projects to be successfully concluded within 10 to 15 percent of the intended timeline, and the final release timelines were significantly closer to the expected timelines. For example, the “Weather App” was a project undertaken by the web development team, with only web developers engaged, and the suggested model seems to function very well in the most recent version v4 of the “Weather App.”

5.3.2 Negative Results

One thing we discovered when integrating our suggested approach into our project is that certain projects take longer than anticipated, like the project “Family locator”. Family Locator Project v3 follows about 80% of our planned model's principles, however the project has nevertheless gone far past its projected timeframe. Despite being a mobile application, the “Family locator” project included two additional programming teams besides the android mobile development team; backend development and database development. We required programmers from the backend and database development teams to work on this project, and assist us hold those user records in a database, on a distant server.

The “programmer” position was suggested for all three teams, despite the fact that their domains were distinct. We noticed that because the project involved multiple teams and domains, there were issues with communication between developers on different teams, and testing took longer than expected because different domains required different types of test cases, and even if one domain passed all the intended tests, the developed functionality of that domain needed to be re-tested when integrating with the other domains.

5.4 Analysis of our Proposed Solutions for Remote Working Environment

The remote work environment is now being adapted worldwide, be it due to the COVID pandemic or the bigger organizations finding it economical to hire developers who are working remotely on their projects. Any team which is well trained in a co-located environment needs a certain number of modifications to the agile model that they are following, along with the necessary training on those practices before the remote working agile model can be implemented in its essence.

After more than a year of working in a remote environment and actively implementing our proposed solutions for the remote working environment as well as developing new softwares using those solutions, our development team, which consisted of nine team members, was asked to vote against some agile practices and agile features, as well as the work modality that they preferred for that agile practice. The results of the voting session are shown in Table 14.

Table 14 - Results of the Agile Team Voting

Agile Features	**Co-located Team**	**Remote Team**
Understanding of Requirements	4	5
Communication Style	6	3
Documentation	2	7
Progress Meetings	1	8
Development Style	3	6
Requirement Changes	4	5
Pair Programming/Code Reviews	0	9
Work Distribution	3	6
Seeking help from senior members	7	2
Daily Meetings	5	4
Quality Assurance Activities	7	2
Focus	1	8
Motivation	5	4
Knowledge Gain	3	6

5.4.1 Discussion and Analysis

Analysis of the results of the questions we asked to the agile team members, it is evident that at some point like code reviews, better understanding of the requirements, focus and knowledge gain, the remote practices we used were the clear winners. However, at some points like seeking help from senior members of the team, motivation and the daily meetings which are slightly more prolonged during a remote setup, to address ambiguities and discuss any blockers that the team is facing, the team members opted for the practices used in the co-located environment.

This indicates us that that one of the challenges that come with working in a distributed and remote setup, is engagement of the team members with other team members, which can be hindered by various factors like members being located in different time zones, having different working hours or at time, even a poor internet connection can also lead to problems where a senior member of the team might not be reachable when his input is required and the junior members may look at it as a blocker to their work and progress. As the remote agile model keep getting used by the agile team, there are various practices that can be introduced like a weekly developers meeting where all developers get together and discuss technology trends, new domain related programming practices and if some developer is facing a major blocker is implementation of some requirements, as this can prove to be very beneficial for the junior team members.

5.5 Summary

We examined and discussed the effects of our suggested solutions for catering to the difficulties of a remote working environment in this chapter, as well as the outcomes obtained when our proposed model was applied in a real-world situation to create new softwares.

Chapter 6

6. Conclusion and Future Works

6.1 Chapter Outline

In this chapter, the thesis is completed with a brief overview of all the work done in this study. The study concludes with a discussion of what results were achieved during this study along with some challenges that were encountered and the research's future path. This research's potential future work is also mentioned.

6.2 Conclusion

Most agile models will fail in a standalone approach for small-scale products with really volatile and unclear requirements, a small scope of development, and the need to deliver a working release to be rolled out to end users, according to the findings and analysis of this research. Some agile models will focus on iterative development leading up to one final product, while there might be a lot of overheads like excessive documentation and requirement engineering in another mode. When we first began developing our projects, we discovered some bottlenecks in the existing models regarding various aspects like the number of face-to-face meetings, the people that make up and agile team, and excessive documentation, particularly when the project was delayed or costing more than it should in terms of financial cost and time. Those gaps were then discussed among the team, and different solutions were proposed, with the best ones being chosen and implemented in the following sprints. As a consequence of continuously experimenting and creatively changing the current model in tiny but solid and well documented steps, the modified agile model has been our method of addressing those loopholes that were identified in the previous models. The modifications could not have been too dramatic, as the members of the team would get disoriented and lose focus, so they were introduced gradually and consistently across our projects.

We can now confidently state that our modified model is stable enough to be used in future projects and will not undergo any significant modifications, despite the fact that it is still a work in progress. There's also clear evidence that our model works really well for small-scale projects with just one

domain, such as mobile, backend, or web and the projects were successfully finished within 10 to 15 percent of the intended timeline, but when multiple domains are involved, our model performs a little better than previous agile models from where we took our inspiration, but the multi-domain project can still go well beyond the intended timeline.

In this research we also discussed the benefits that come with using agile methodologies for developing our softwares and what challenges can arise when the work modality is shifted from in-house to remote work and then we discussed some of the modifications that we made to the existing agile model to cater the new remote work modality along with the steps which are required for such a model to be efficiently implemented for a team. The modifications that were discussed were implemented in a real-world scenario and the agile team followed the modified agile model for over a year. Analysis of the agile model was conducted and it was found out that overall the modified agile model was successfully implemented and for some agile practices, the team preferred the remote model over the co-located agile model and we also had a clear indication that remote working and centralized information approach helped the team to get better understanding of the project and the requirements.

6.3 Future Works

In future, we'll need to put our proposed model through its paces in medium- to large-scale projects to see how well it works. So far, our system has been utilized with a team where some members are co-located and some work remotely. It will be fascinating to observe how our system performs if the majority or all of the team is spread over several time zones and working remotely. There's also evidence that our model works really well for small-scale projects with just one domain, such as mobile, backend, or web, but when multiple domains are involved, our model performs a little better, but the multi-domain project can still go well beyond the intended timeline. More work needs to be done to address these issues in the future, where we can work on introducing processes that can help with these issues.

We have also observed that at some of the points like seeking help from individual team members and long online progress meetings the team preferred the co-located agile model. In future, we can look at making certain modifications to enhance the engagements in online meetings and to improve the collaborations between individual team members.

6.4 Summary

We wrapped up the research work in this chapter by discussing all of the suggested work and evaluating it. This chapter completed the discussion of both the suggested agile model and the proposed solutions for adapting the agile model in a remote working environment. We also outlined the difficulties that were encountered, as well as a road map for future study that outlined all of the potential connected to this thesis.

REFERENCES

[1] Sherrell, L., 2013. Waterfall Model. Encyclopedia of Sciences and Religions, pp.2343-2344.

[2] Beynon-Davies, Paul & Carne, C & Mackay, Hugh & Tudhope, D. (1999). Rapid application development (RAD): An empirical review. European Journal of Information Systems. 8. 10.1057/palgrave.ejis.3000325.

[3] Schwaber, K., 1997. SCRUM Development Process. Business Object Design and Implementation, pp.117-134.

[4] Beck, K., Andres, C. and Kado, M., 1996. Extreme Programming Explained: Embrace Change.

[5] Beck, K.; Beedle, M.; van Bennekum, A.; Cockburn, A.; Cunningham, W.; Fowler, M.; Grenning, J.; Highsmith, J.; Hunt, A.; Jeffries, R.; Kern, J.; Marick, B.; Martin, R. C.; Mellor, S.; Schwaber, K.; Sutherland, J. \& Thomas, D. (2001), 'Manifesto for Agile Software Development' 'Manifesto for Agile Software Development'.

[6] Brooks, Jr, Frederick. (1995). The Mythical Man-Month: Essays on Software Engineering.
[7] Sommerville, I.: Software Engineering, 7th edn. Pearson Eductation Ltd., London (2004)

[8] McBreen, P.: Software craftsmanship: the new imperative. Addison-Wesley, Boston (2002)

[9] Jarzombek, J.: The 5th annual jaws s3 proceedings (1999) [1] Sherrell L. (2013) Waterfall
[10] Ken Schwaber and Mike Beedle. 2001. Agile Software Development with Scrum (1st. ed.). Prentice Hall PTR, USA.
[11] C, John. (2020). Blended Agile Approach: A Mobile Application Approach. International Journal of Advanced Trends in Computer Science and Engineering. 9. 465-469. 10.30534/ijatcse/2020/7391.32020.

[12] Sanchez, Jacqueline. (2018). Challenges of Traditional and Agile Software Processes.
[13] Anwer, Faiza & Aftab, Shabib & Muhammad, Syed & Waheed, Usman. (2017). Comparative Analysis of Two Popular Agile Process Models: Extreme Programming and Scrum. International Journal of Computer Science and Telecommunications. 8. 1-7.

[14] Abrahamsson, Pekka & Koskela, J. (2004). Extreme programming: A survey of empirical data from a controlled case study. Proceedings - 2004 International Symposium on Empirical Software Engineering, ISESE 2004. 73- 82. 10.1109/ISESE.2004.1334895.
[15] Fojtik, Rostislav. (2011). Extreme Programming in development of specific software. Procedia CS. 3. 1464-1468. 10.1016/j.procs.2011.01.032.
[16] Singh, Harshpreet. (2014). FDRD: Feature Driven Reuse Development Process Model. 10.1109/ICACCCT.2014.7019376.
[17] Anwer, Faiza & Aftab, Shabib. (2017). SXP: Simplified Extreme Programing Process Model. International Journal of Modern Education and Computer Science. 9. 25-31. 10.5815/ijmecs.2017.06.04.
[18] Abdullahi Wakili, Almustapha & Alhasan, Lawan & Kamagata Hamisu, Abubakar. (2020). QUALITY ASSURANCE PRACTICES IN AGILE METHODOLOGY.

[19] Khan, Rizwan \& Srivastava, Akhilesh \& Pandey, Dilkeshwar. (2016). Agile approach for Software Testing process. 3-6. 10.1109/SYSMART.2016.7894479.

[20] Shubhangi, A.Deshpande & Deshpande, Anand & Marathe, Manisha & Garje, Goraksh. (2010). Improving Software Quality with Agile Testing. International Journal of Computer Applications. 1. 10.5120/440-673.

[21] Itkonen, Juha & Rautiainen, Kristian & Lassenius, Casper. (2005). Towards Understanding Quality Assurance in Agile Software Development. 8.
[22] Abrahamsson, P., et al. 2003. "New Directions on Agile Methods: A Comparative Analysis," Proceedings of 25th International Conference on Software Engineering, pp. 244-254.

[23] Elssamadisy, A., G. Schalliol 2002. "Recognizing and Responding to "Bad Smells" in Extreme Programming," Proceedings of the 24th International Conference on Software Engineering, pp. 617-622.

[24] Lippert, M., et al. 2003. "Developing Complex Projects Using XP with Extensions," Computer, 36 (6): 67-73
[25] Stephens, M., D. Rosenberg 2003. Extreme Programming Refactored: The Case Against XP. Berkeley: Apress.

[26] Theunissen, W.H.M., D.G. Kourie, B.W. Watson 2003. "Standards and Agile Software Development," Proceedings of SAICSIT, pp. 178-188.

[27] Jurca, Gabriela & Hellmann, Theodore & Maurer, Frank. (2014). Integrating Agile and User-Centered Design: A Systematic Mapping and Review of Evaluation and Validation Studies of Agile-UX. Proceedings - 2014 Agile Conference, AGILE 2014. 24-32. 10.1109/AGILE.2014.17.

[28] Nawaz, Zahid & Aftab, Shabib & Anwer, Faiza. (2017). Simplified FDD Process Model. International Journal of Modern Education and Computer Science. 9. 53-59. 10.5815/ijmecs.2017.09.06.

[29- Saleh] Saleh, Sabbir & Huq, Syed & Rahman, Mohammed. (2019). Comparative Study within Scrum, Kanban, XP Focused on Their Practices. 10.1109/ECACE.2019.8679334.

[30] Rychly, Marek & Ticha, Pavlina. (2007). A Tool for Supporting Feature\-Driven Development. 5082. 196-207. 10.1007\/978-3-540-85279-7_16.

[31- Yasvi] Yasvi, Maleeha. (2019). Review On Extreme Programming-XP.

[32] Aftab, Shabib \& Nawaz, Zahid & Anwar, Madiha & Anwer, Faiza & Bashir, Salman \& Ahmad, Munir. (2018). Comparative Analysis of FDD and SFDD. International Journal of Computer Science and Network Security. 18. 63-70.

[33] Calderón-Martínez, Olga & Rojas, Alix & Mejía, Camilo. (2020). A Scrum Implementation Plan by Phases and Oriented by Team Members: A Case Study.

[34] Dingsøyr, Torgeir & Dybå, Tore & Moe, Nils. (2010). Agile Software Development: An Introduction and Overview. Agile Software Development, by Dingsøyr, Torgeir; Dybå, Tore; Moe, Nils Brede, ISBN 978-3-642-12574-4. Springer-Verlag Berlin Heidelberg, 2010, p. 1. -1. 1. 10.1007/978-3-642-12575-1_1.

[35] Hj.diva-portal.org. 2021. How Covid-19 and working from home have affected agile software development. [online] Available at: <https://hj.diva portal.org/smash/get/diva2:1576505/FULLTEXT01.pdf> [Accessed 3 October 2021].

[36] Shrivastava, Suprika & Date, Hema. (2010). Distributed Agile Software Development: A Review. J Comput Sci Eng. 1.

[37] Kude, T., Mithas, S., Schmidt, C., & Heinzl, A. (2019). How pair programming influences team performance: The role of backup behavior, shared mental models, and task novelty. Information Systems Research 30.4, 1145-1163.

[38] Schwaber, K., & Beedle, M. (2002). Agile Software Development with Scrum (Vol. 1). Upper Saddle River: Prentice Hall.

[39] da Camara, R., Marinho, M., Sampaio, S., & Cadete, S. (2020). How do Agile Software Startups deal with uncertainties by Covid-19 pandemic. arXiv preprint arXiv:2006.13715

[40] Android. 2021. Android – Certified - Partners. [online] Available at: <https://www.android.com/certified/partners/>.

[41] GitHub Docs. 2021. About project boards - GitHub Docs. [online] Available at: https://docs.github.com/en/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards

[42] Firebase Analytics For Android. 2021. Firebase. [online] Available at: https://firebase.google.com/?gclid=CjwKCAjwqeWKBhBFEiwABo_XBs2AAZkD69iAo9VC0ZnPxLKxBKZUH3-N6mn6kqbT4mF6yUp6xbCSVhoCKXsQAvD_BwE&gclsrc=aw.ds

[43] Carvalho, Bernardo & Henrique, Carlos & Mello, Carlos. (2011). Scrum agile product development method -literature review, analysis and classification. Product: Management & Development. 9. 39-49. 10.4322/pmd.2011.005.

[44] GitHub Docs. 2021. About pull requests - GitHub Docs. [online] Available at: <https://docs.github.com/en/github/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>