



The adoption of Software Engineering practices in a Scrum environment

Oluwaseun Alexander Dada & Ismaila Temitayo Sanusi

To cite this article: Oluwaseun Alexander Dada & Ismaila Temitayo Sanusi (2021): The adoption of Software Engineering practices in a Scrum environment, African Journal of Science, Technology, Innovation and Development, DOI: [10.1080/20421338.2021.1955431](https://doi.org/10.1080/20421338.2021.1955431)

To link to this article: <https://doi.org/10.1080/20421338.2021.1955431>



© 2021 The Author(s). Co-published by NISC Pty (Ltd) and Informa UK Limited, trading as Taylor & Francis Group



Published online: 21 Aug 2021.



Submit your article to this journal [↗](#)



Article views: 68




View related articles [↗](#)



View Crossmark data [↗](#)

The adoption of Software Engineering practices in a Scrum environment

Oluwaseun Alexander Dada^{1,2*} and Ismaila Temitayo Sanusi ³

¹*Institute for Molecular Medicine Finland (FIMM), University of Helsinki, Helsinki, Finland*

²*The School of Software, Lekki-Lagos, Nigeria*

³*School of Computing, University of Eastern Finland, Joensuu, Finland*

*Corresponding author. Email: alexander.dada@helsinki.fi, alexander.dada@schoolofsoftware.net

The competition in the software market demands that the time required for any software product to reach the market be reduced if the product is to survive competition from other developers. The pursuit of this goal has led to the adoption of agile software development methodologies. While other agile methodologies provide guidelines as to the software engineering (SE) practices to be used during the development lifecycle, Scrum does not. The purpose of this study is twofold: first, to identify the usage and level of importance of software engineering practices in the Scrum development environment; and second, to investigate how Scrum teams adopt an appropriate set of SE techniques and whether a hybrid Scrum/Extreme Programming (XP) methodology is an appropriate approach to take. This research was conducted by examining sample data from five organizations using the Scrum methodology. The sample included a range of industries including communications and embedded systems, financial asset management, software development houses and consulting firms in South Africa. The study employed a mixed method approach. A key finding was that, regardless of the fact that Scrum does not explicitly recommend engineering practices, there was extensive use of these practices by all of the participating organizations. The study also found that the lack of software engineering practices in Scrum does not constitute a barrier to a successful adoption of Scrum, provided the ‘inspect and adapt’ principle inherent in Scrum is properly followed. The study discusses the findings, explains the implications and suggests future research.

Keywords: Scrum, extreme programming, software engineering, agile management

Introduction

Agile software development has moved from the fringes of the software development community to the mainstream. This movement is driven by the need to produce better software faster, which is integral to developing competitive advantage in the global software community. From North America to Asia and everywhere in between, the ability to deliver software that delights the customer has become a critical success factor (Eckstein 2013). There are a number of agile methodologies, but the following are the more important ones: Extreme Programming (XP), Adaptive Software Development (ASP), Dynamic Systems Development Method (DSDM), Scrum, Crystal, Feature Driven Development, and Agile Modeling. They all follow the core values and principles defined in the agile manifesto (Lei et al. 2017; Srivastava, Bhardwaj, and Saraswat 2017).

Agile methodologies usually contain practices that focus on both dimensions of software development: Project Management and Software Engineering (SE) (Jyothi and Rao 2011). Project Management practices concentrate on planning, organizing, securing and managing resources to bring about the successful completion of the system development project and the realization of its objectives. On the other hand, Software Engineering practices focus on the technical side of the development practice like coding, refactoring, pair programming and other tools and techniques used in the analysis, design, construction and testing phases of the project (Jyothi and Rao 2011; Brhel et al. 2015). While most of the agile methodologies focus on both software engineering and project

management, Scrum is predominantly a project management methodology (Jyothi and Rao 2011). Consequently, while other agile methodologies provide guidelines as to the software engineering practices to use during the development lifecycle, Scrum does not (Abrahamsson and Salo 2008; Jyothi and Rao 2011).

The purpose of this empirical research study is twofold. Firstly, the study will attempt to identify the level of usage and importance of software engineering tools and techniques used by Scrum teams. The second objective is to investigate how Scrum teams adopt an appropriate set of SE techniques and whether a hybrid Scrum/Extreme Programming (XP) methodology is an appropriate approach to take. This study is of relevance considering the rate at which the Scrum methodology is being adopted globally. According to Pudusserry (2009), 37% of all agile implementations are in Scrum while a further 23% use a combination of Scrum and Extreme Programming. For instance, another study shows that Scrum was the most utilized approach in a systematic review conducted from 2010 to 2016 (Vallon et al. 2018). In addition, the study findings should provide guidance in the selection of appropriate software engineering tools and techniques for a software development team migrating from the traditional methodologies to a Scrum setting.

Based on the objectives, this study seeks to provide answers to the following research questions:

RQ1. What is the level of usage and importance of software engineering practices in the Scrum development environment?

RQ2. How do Scrum teams adopt an appropriate set of SE techniques (trial and error, via coaching support or other means)?

RQ3. Is the adoption of a hybrid Scrum/Extreme Programming methodology an appropriate approach to take?

The next section reports related research, while the section following that reports the methodological process adopted. The section thereafter presents the analysis and findings of the study followed by the section that discusses the results and implications while the final section presents the conclusion, the limitations that arose during the study and offers suggestion for future research.

Literature review

Software has been part of modern society for more than 50 years (Awad 2005). There are various types of software development methodologies in use today, but according to Kanwal, Junaid, and Fahiem (2010), the most popular kinds of methodologies are the traditional software development methods and a group of iterative methods, including the family of agile methodologies (Figure 1).

Traditional software development methodology

The traditional method of software development, often referred to as the waterfall approach, assumes that the process of building a software system follows a series of sequential steps. Each of the phases consists of a fixed set of practices and deliverables that must be accomplished before the following phase can begin. The phases may be named differently (Awad 2005), but the logic is that the first phase involves planning and capturing of the system's requirements and the second phase determines how the system should be designed to meet these requirements. The third stage is where the developers write the code and implement the system, followed by the verification phase to ensure that the system performs as specified. Finally, the maintenance phase focuses on issues related to the system that arise after deployment (Awad 2005; Pressman 2005).

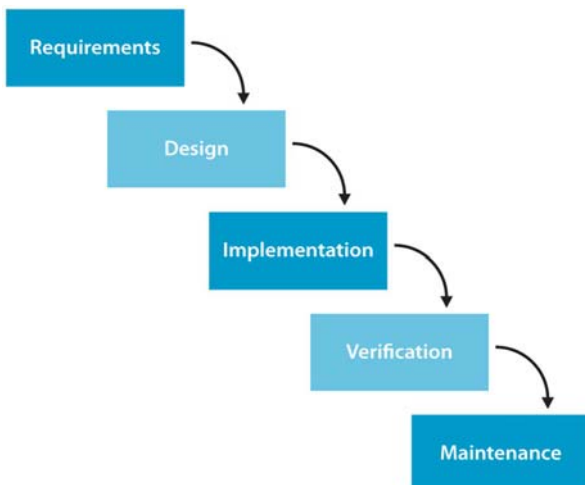


Figure 1: Waterfall methodology (FreeTutes 2011).

Agile approach

The agile approach refers to a family of development techniques designed to deliver products on time, on budget, with high quality and customer satisfaction (Jyothi and Rao 2011). The agile methods attempt to focus on the primary goals of the software development process, that is, the creation of working (defect-free) software (Hunt 2006). One of the ways in which agile methodologies effectively manage change is through dynamic prioritization, meaning ‘at the end of every iteration, the customer can reprioritize the features desired in the next cycle, discarding originally planned features and adding new ones’ (Highsmith and Cockburn 2007, 3). XP uses story cards; Scrum uses the term ‘backlog’; Agile Software Development (ASD) and Feature-Driven Development refer to features (Pressman 2005; Highsmith and Cockburn 2007).

The agile methodologies are all based on iterative development. In addition, Pressman (2005) reasons that agility can be applied to any software process provided that the development team is allowed to streamline tasks and conduct planning in a way that eliminates non-essential work products. It is also interesting to note that most agile methodologies cover both software engineering and project management aspects of the software development process. In contrast, the Scrum methodology is predominantly a project management technique which gives no guidance on how the software engineering practices are to be implemented (Srivastava, Bhardwaj, and Saraswat 2017). This is perhaps the reason why Appelo (2008) says Scrum by itself is never enough, and that development teams must adopt additional practices (usually XP).

The various tools and techniques that form part of a typical software development process may be classified into two groups: project management (PM) and software engineering (SE) practices (Appelo 2008). While PM focuses on the planning and guiding of the project processes and resources from start to finish, SE deals with the various practices used in the building of the system (Kerzner 2009). The traditional PM approach is often referred to as ‘*plan based*’ or ‘*predictive*’ while the agile approach is referred to as ‘*empirical*’ or ‘*adaptive*’ (Keshta and Morgan 2017). Lynema (2010) and Highsmith and Highsmith (2009) highlight some of the common principles and practices encapsulated in the traditional project management approach in contrast to those followed in agile project management (Table 1).

The Scrum methodology

Scrum can simply be defined as an agile software development methodology that is based on multiple small teams working in an intensive and interdependent manner. The term ‘Scrum’ is named after the scrum (or scrummage) formation in rugby, which is used to restart the game after an event that causes play to stop, such as an infringement (Stafford et al. 2011).

As illustrated by Figure 2, a Scrum project usually starts with a vision of the system to be developed. Then a list of all the user requirements (that are currently known) is developed. This list is referred to as the Product Backlog (PB). The PB is prioritized and

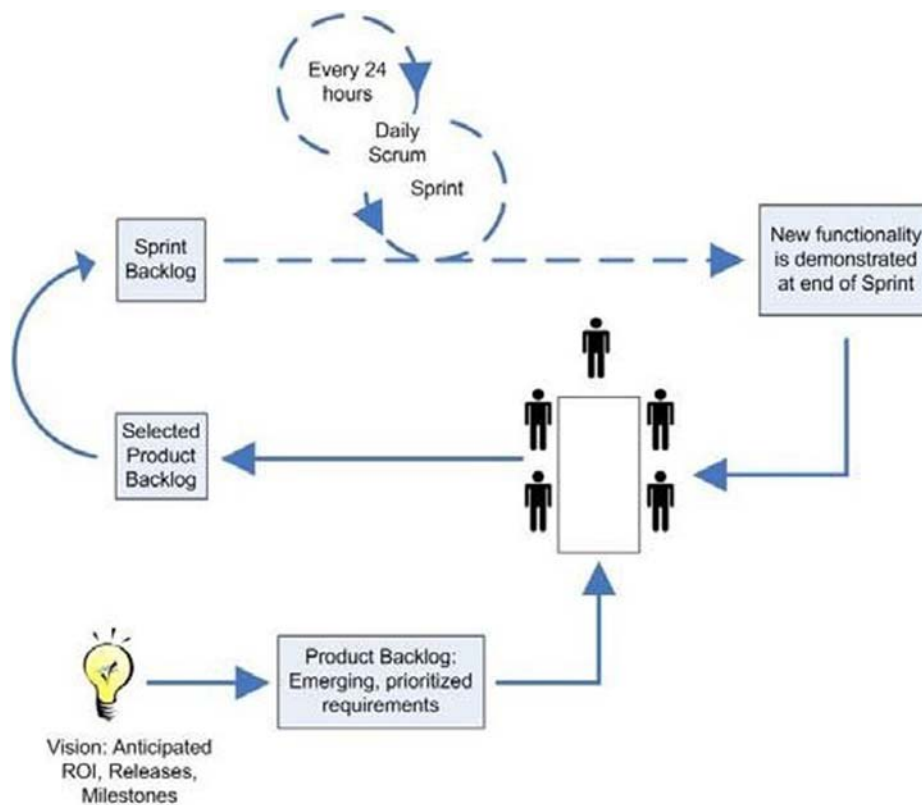


Figure 2. Detailed Scrum flow (Schwaber 2004).

divided into proposed releases. All work is done in Sprints (each Sprint is an iteration, of between 2 and 4 weeks). Every Sprint starts with a Sprint planning meeting, where the Product Owner (user representative) and the team meet to discuss what must be done for the next Sprint. This translates into the next Sprint Backlog (a list of tasks that must be performed to deliver a completed increment of potentially shippable product functionality by the end of the Sprint). The Team holds Scrum meetings on a regular basis (usually daily) for 15-minutes. The purpose of the meeting is to synchronize the work of all team members and to schedule any meetings that the team needs to progress. At the end of the Sprint, the team gets together for a Sprint review meeting in which the team demonstrates to the Product Owner what was developed during the Sprint. After the Sprint review the Scrum Master (team facilitator, coach and mentor) calls for a Sprint retrospective in order to encourage the team to reflect on their current development process and practices and, where appropriate, to adapt these to make the team more effective and efficient for the next Sprint (Drnovscek and Mahnic 2005; Pressman 2005).

From the above description it is clear that Scrum ultimately focuses on project management issues – such as promotion of self-directed teams, and daily team measurement. However, Scrum does not prescribe any software engineering practices (Larman 2004; Pudusserry 2009). While Appelo (2008) observes that Scrum is fast becoming a de facto standard for managing agile software development projects, he concludes that Scrum must be augmented by another methodology that can supply the

engineering practices critical to developing quality software.

Scrum and extreme programming (XP)

Sahota (2011) argues that Scrum contains a set of practices (such as self-organizing and cross functional teams, retrospective, and many others) that can enhance project management practices of the Extreme Programming (XP) methodology. Sahota (2011) and Appelo (2008) note that relevant Scrum practices need to be added to XP to ensure a successful software project, although there are practices that overlap between the two agile methods (the idea is illustrated in Figure 3). In agreement with Sahota's (2011) argument; Appelo (2008) claims that the Scrum framework is intentionally incomplete and also stresses that Scrum is originally aimed at providing project management techniques for other agile methods such as XP, ASP, DSDM.

While Scrum is focused on project management, there is little argument concerning the requirement that team members must adopt appropriate software engineering practices to ensure they build quality systems (Fowler et al. 2002). Cohn (2011) states that a good Scrum team will find appropriate technical practices to ensure that they produce high quality software, while a bad Scrum team (if they can be said to be doing Scrum) will use Scrum as an excuse to do low quality work. Cho (2009), Kim (2007) and Pudusserry (2009) also observe that Scrum does not recommend any particular practices, as different development environments require different tools and techniques. Schwaber and Beedle (2002) recommend the use of XP practices such as pair

Valuable Scrum Practices Exist Outside XP

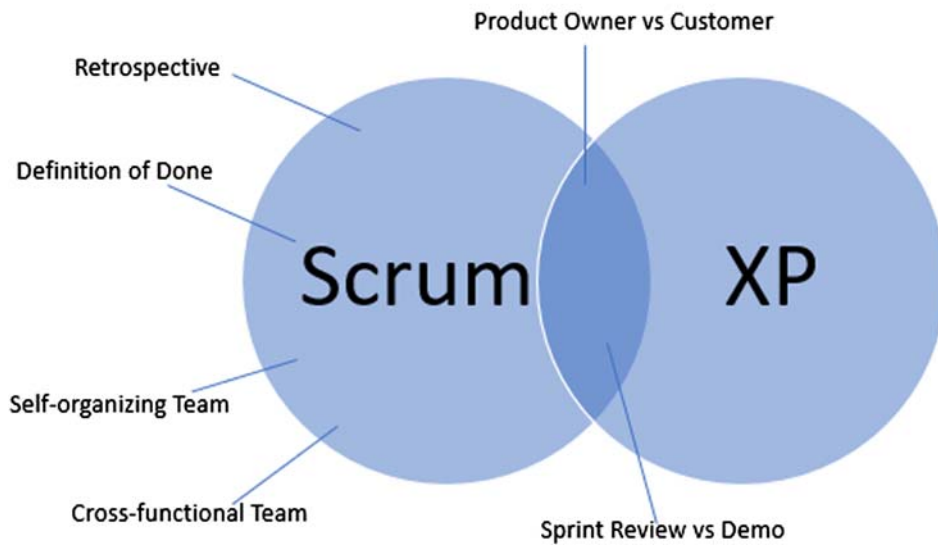


Figure 3: Scrum contains valuable practices that lie outside XP (Sahota 2011).

programming, collective code ownership and minimal documentation. On the other hand, Cho (2009) suggests that the engineering techniques of the Rational Unified Process (RUP) be adopted in Scrum and Sutherland (2007) endorses the use of Lean tools and techniques to complement the project management processes of Scrum.

Shore and Warden (2008) cross reference agile practices that are similar (not identical) in both Scrum and XP ranging across the following five categories: Thinking, Collaborating, Releasing, Planning, and Developing. The outcome shows that Scrum covers 53% of the defined practices compared to XP’s 90% coverage. This finding supports Appelo’s (2008) and Sahota’s (2011) observation that XP is almost complete in itself while Scrum is not. In terms of implementation and testing, XP catered for 89% of the practices investigated against Scrum’s 11%, thus confirming the lack of engineering techniques in Scrum. In a similar survey conducted by Pudusserry (2009), the

results indicated that 37% of the respondents were Scrum users, 12% used XP, while 23% used a Scrum/hybrid approach. The author concludes by recommending that the best way to successfully implement Scrum ‘is to start with Scrum (that is, focus on the management practices) and then adopt the engineering practices from XP’ (Pudusserry 2009, 8). In the context of this discussion, it is appropriate to reverse the argument put forward by Sahota (2011) on how Scrum can enhance XP, and rather ask the question, how can XP enhance Scrum?

Figure 4 illustrates how many of the popular SE practices that are recommended in XP and other agile methodologies can be used to enhance the Scrum development environment.

Summary of the literature

The pressures of a truly global economy cause today’s businesses to increasingly rely on their ability to

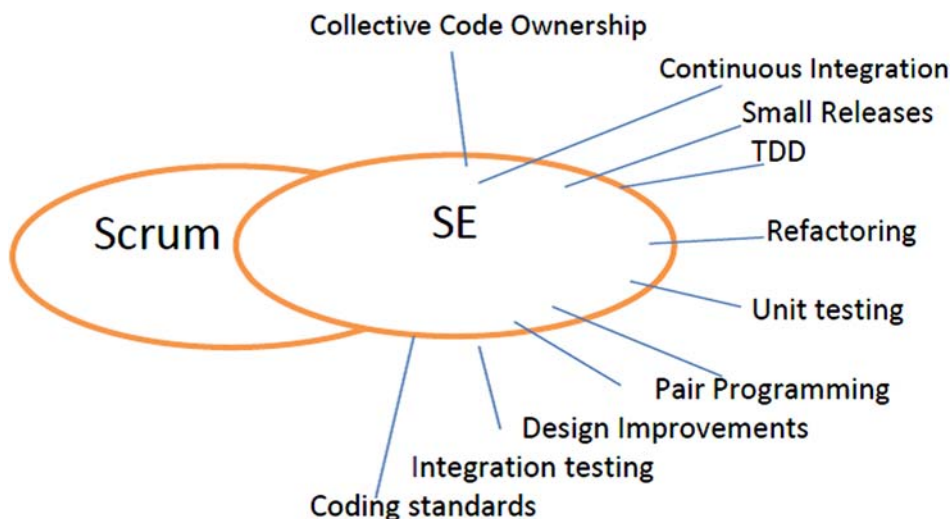


Figure 4: Enhancing Scrum with XP Software Engineering Practices (Pressman 2005; Appelo 2008; Cho 2009; Sahota 2011).

produce quality software to remain competitive, whether it be software for managing manufacturing and customer delivery processes or software for improving the efficiency of day-to-day activities. As a result, various development methodologies have been explored, ranging from the so-called Waterfall methodology to the agile approach. While the Waterfall approach serves as an improvement over previous development methods, its inability to easily respond to changes (among other limitations) has led to the emergence of the agile software development methodologies (Bang 2007; Dyba and Dingsoyr 2008).

According to the literature reviewed, the Scrum methodology is arguably the most adopted agile approach. Scrum focuses on the effective management of the software project by promoting better communication channels between the development team and the stakeholders. The Scrum method also makes it easy for the owners of the system to reprioritize their desired features without interrupting the development process. The short sprints that Scrum advocates make it easy to get quick feedback from the project owners, and at the same time it is easy for the development team to cope with changes. Scrum also makes it possible for management to easily track the productivity of each of the team members. The most important benefit of the Scrum methodology is that it increases team productivity, software quality, and user satisfaction (Bjeirmi and Munns 1996). However, Scrum is a project management framework and lacks guidance on appropriate software engineering practices (Cao and Ramesh 2008).

As a result, various authors Appelo (2008), Kanwal, Junaid, and Fahiem (2010) as well as Pudusserry (2009) and practitioners (Sutherland (2004), Cohn (2011), and Cockburn (2001)) suggest that Scrum management practices should be complemented by software engineering tools and techniques from other agile methodologies. According to Pudusserry (2009), it is up to the team to decide which ones to use. Considering the fact that very little empirical research has been conducted regarding the usage and adoption of software engineer practices within the Scrum environment a number of questions are posed.

Given the lack of formal guidance from the Scrum methodology:

- What software engineering practices are commonly adopted by Scrum developers?
- How do Scrum teams identify and implement an appropriate set of SE practices – by prescription or discovery?

These two questions encapsulate the main research objectives of this study.

Methodology

Very few research studies have been conducted specifically to identify the software engineering tools and techniques that are used by software development teams that have migrated from a traditional development approach to Scrum. As a result, a mixed methodology research approach was adopted for comprehensive investigation into these research objectives. The empirical research

conducted was exploratory in nature. The quantitative approach was used to identify which software engineering practices are currently used in the Scrum development environment. The instrument used to achieve this objective was a questionnaire which was directed at Scrum team members and Scrum Masters (team facilitators) in the sample companies. On the other hand, the qualitative aspect of the research was targeted at experienced practitioners in the Scrum community and took the form of semi structured interviews. The purpose of the interviews was to elicit expert opinion on how software engineering practices are adopted.

Research sample

The data for the research was collected from software development teams that have changed from a traditional software development methodology to Scrum. Mixed methodology approach was utilized in this study to understand how the software engineering practices are adopted. This involves the use of quantitative and qualitative data derived from a questionnaire and interview schedule. Scrum community is the targeted population of the study. Purposeful sampling technique was employed to recruit Scrum team members, Scrum Masters and experienced practitioners in the sample companies. Before adopting purposeful sampling technique, the researcher decides what needs to be known and sets out to find people who can and are willing to provide the information by virtue of knowledge or experience (Etikan, Musa, and Alkassim 2016). In all, thirty-five participants responded to the questionnaire and nine participants were interviewed. The main participants in this research were the companies and individuals in the Cape Town and Stellenbosch Scrum community in South Africa. These companies can be categorized into the following industries: communications and embedded systems, financial asset management, software development and IT consulting.

Data collection

Questionnaires

The purpose of the questionnaire was to identify the usage and importance of software engineering practices in the Scrum community. The questionnaire begins with a brief introduction to the research team and to the purpose of the study. It then asks for demographic information including the respondent's last role in a traditional methodology team, their role(s) in the Scrum team, and number of years of experience in both teams. Participants were requested to tick only the practices that they have used, while others were left blank. If the respondent had used certain techniques other than the ones mentioned in the questionnaire, they were encouraged to enter such practices in the extra columns provided. Finally, the participants were asked to state any problem(s) they might have experienced while migrating from the traditional approach to the Scrum environment in relation to the selection and adoption of software engineering tools and techniques. The authors were able to restrict the questionnaire to two pages in the hope and anticipation that this would encourage participants to complete it timeously.

Table 1: Comparison of principles and practices in traditional and agile project management (Highsmith and Highsmith 2009; Lynema 2010).

Items	Traditional project management	Agile project management
Planning	Plan all in advance	Plan as you go
Artefact	Work breakdown structure	Feature breakdown structure
Requirements gathering	Functional specification	User story
Project schedule	Gantt chart	Release plan
Communication tool	Status reports	Story boards
Development model	Deliver at the end	Deliver as you go
Continuous learning	Learn at the end	Learn every iteration
Adaptability	Follow the plan	Adapt everything
Management	Manage tasks	Manage team

Table 2: Interview respondents.

Code	Role and responsibility
Managers (M1 to 4)	System development or operation managers with responsibility for a number of Scrum teams
Developers (D1)	Team member in a Scrum team
Scrum masters (SM1 to 3)	Team manager, facilitator and coach. Looks after Scrum team(s) and shields them from external disturbance
Coaches (C1)	Gives training to Scrum team(s) in areas where they need coaching and assistance

The items in the questionnaire were derived from the various practices (see Tables 1 and 2) put forward by various authors and ‘agilists’ such as Schwaber and Beedle (2002), Sutherland (2007), Fowler et al. (2002), Cohn (2011) as well as Sahota (2011). A five-point Likert-type scale was used in the questionnaire ranging from (1) ‘*Very Unimportant*’ to (5) ‘*Very Important*’. This allowed users a range of possible answers, enabling them to rate each practice by its level of importance to the development process.

Semi structured interviews

The purpose of the interviews was to ask questions around the second research objective. This helped the authors to understand how the Scrum team identified and adopted the engineering tools and techniques they were currently using and if there were any issues relating to the selection of the appropriate practices. To achieve these goals, six questions were prepared. Nine suitable interviewees (experienced Scrum professionals) were identified, and

emails were sent to solicit their support. Interviews were set up and individual members of the team visited the participants in their respective organizations. The interviews took between 30 and 60 min. One long distance interview was conducted by telephone. In each of the interview sessions, the research team explicitly sought the consent of the interviewees to be recorded.

As the interviewee’s role in the Scrum process could influence their opinions on and attitudes to issues raised in the interview, they were categorized as Managers (M), Scrum Masters (SM), Developers (D) and Coaches (C). The code was followed by a number to identify a particular respondent while keeping their anonymity; the four managers interviewed are represented as M1, M2, M3 and M4 while the three Scrum Masters are denoted as SM1, SM2 and SM3. Since there was only one developer and one coach, they are represented as D1 and C1, respectively. Table 3 describes the roles and responsibilities of these experts at their respective organizations.

Table 3: The most important software engineering tools and techniques used in Scrum.

Ranking	Important engineering practices	Average level of importance (LOI)	SDLC phases
1	User stories	4.5	Analysis
2	Integration testing	4.45	Construction
3	Unit testing	4.43	Construction
4	Continuous integration	4.42	Construction
5	Version control	4.39	Quality Ass
6	Test cases	4.39	Construction
7	Systems testing	4.38	Construction
8	Product backlog	4.35	Analysis
9	Self-documenting code	4.32	Construction
10	Retrospective/reflection	4.26	Quality Ass
11	Walkthroughs/ Reviews	4.16	Quality Ass
12	Secure coding standards	4.14	Construction
13	User acceptance testing	4.13	Construction
14	Refactoring	4.1	Quality Ass
15	Test Driven Development (TDD)	4.08	Construction
16	Test automation	4.08	Construction

Data analysis

This section presents the analysis performed in this study. In the quantitative section, the reliability and construct validity of the measuring items were tested. The values of composite reliability and average variance extracted (AVE) were greater than 0.7 and 0.5, respectively for convergent validity which is tandem with Olaleye et al. (2020a) and Olaleye, Sanusi, and Salo (2020b). Both descriptive and inferential statistics were utilized which include frequency, percentages, mean, standard deviation (SD) and t-test. For the qualitative aspect of the study, thematic analysis and deductive coding technique were adopted for the study. Thematic analysis was described by Braun and Clarke (2006) as a method for identifying, analyzing and reporting patterns within data.

Result

Quantitative results

Respondent demographics

The following demographics provide a profile of the survey sample that participated in this empirical research. The questionnaire was completed by 35 respondents. Of these, one questionnaire was found to be incorrectly completed and excluded from the study.

Roles in Scrum and years of experience: The majority of the respondents (see Figure 5) were either Scrum Masters ($n = 10$, 31% of sample) or team members ($n = 22$, 66% of sample). These roles are appropriate to the study as both are heavily involved in the selection and implementation of the team’s SE practices. There was one Coach in the sample who was also included. However, two of the respondents were product owners and their questionnaires were excluded from the sample as their roles are related more to the business than software development. As a result, 32 questionnaires were used in the study.

Figure 6 illustrates the number of years of working experience that the respondents have had in the Scrum

development environment. Twenty-eight (88%) of the respondents have been working in the Scrum environment for between 1 and 4 years. Only 5 (13%) of the respondents are relatively new to the Scrum environment, and the remaining 4 (13%) have over 5 years’ experience in Scrum methodology. The average years of Scrum experience is just over 2 years, which is a source of concern as this means that, on average, knowledge on issues relating to Scrum is limited. This is not a surprise; given that Scrum is a relatively new methodology. With that said, it is important to mention that the respondents have been in the software development (SD) industry for an average of 8 years which implies that there is a fair amount of SD experience in the group.

One of the key objectives of the study was to investigate the level of usage of SE practices within the Scrum environment. In an attempt to put the level of usage in context, it was compared to the level of usage of SE practices in the traditional environment. Usage statistics for traditional and Scrum practices were identified from the questionnaire in the following way:

- Where respondents gave a level of importance to a practice, this indicated that they had used the practice. This assumption was based on the instructions clearly given in the questionnaire:

Please indicate (with a X) if you used any of the following tools and techniques as part of your previous traditional development environment and/or current Scrum environment and put a cross in the appropriate box that depicts its level of importance. Leave the row blank if you did not use that tool or technique. If you have only been involved in Scrum development, leave the traditional usage section blank.

- According to the data collected, five of the questionnaire respondents had only worked in a Scrum environment, making the total number of respondents for the traditional approach ($n = 27$), against the number for Scrum ($n = 32$).
- A count was made of all respondents using SE practices in the two development environments and these counts

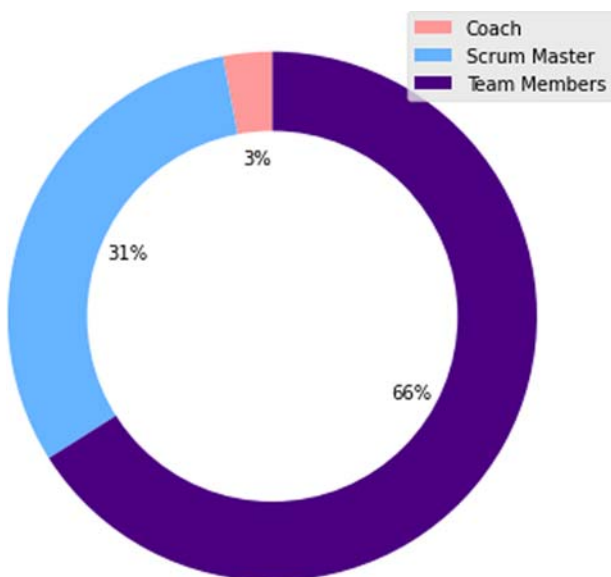


Figure 5: Role of the respondents in Scrum.

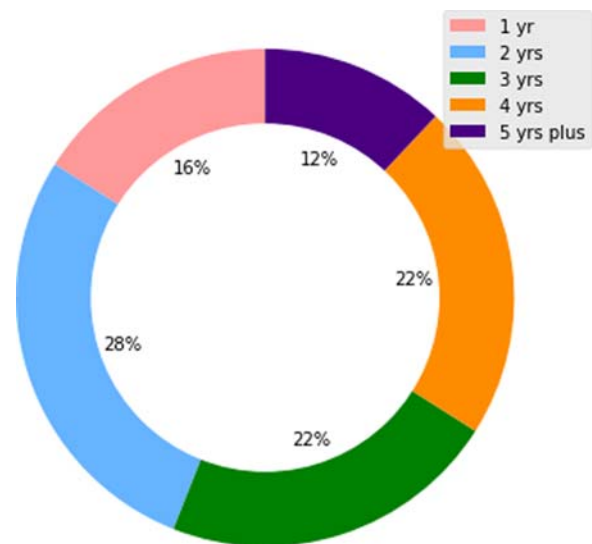


Figure 6: Years of Scrum experience.

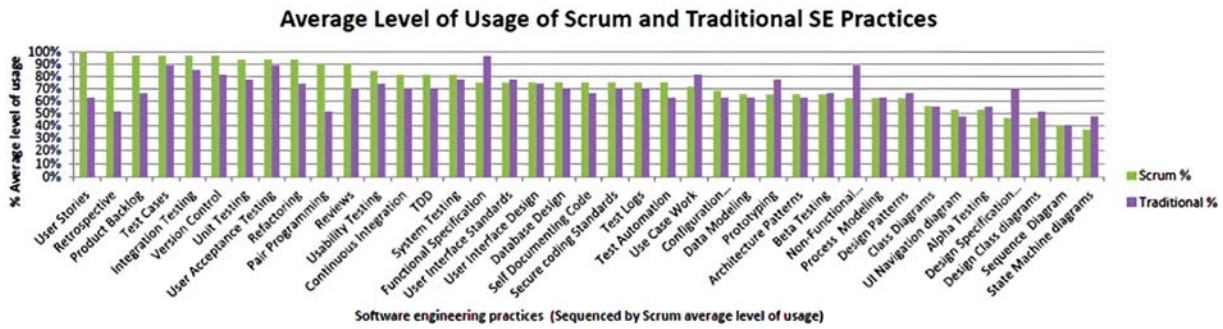


Figure 7: Comparing usage of SE practices in Scrum and traditional methods.

were converted to a percentage usage figure given that the number of responses differed across the two environments.

Figure 7 compares the average-usage of each practice, while Figure 8 shows differences in the usage of the SE practices in both Scrum and traditional environments. The following details some of the important findings derived from the charts:

- Both charts suggest there is more usage of SE practices in Scrum than in traditional methodologies. This is confirmed by the overall average level of usage of SE practices in Scrum (74%) against 69% in the traditional approach.
- Most of the SE practices in Scrum with high levels of usage (percentage usage more than 80%) are construction, testing and quality assurance practices. Given the complexity of the iterative approach and the emphasis of agility on quality, this finding was expected.
- Two practices from the requirements phase (user stories (100% usage) and the product backlog (97% usage)) were among the most used SE Scrum practices. This again was expected as they are key artifacts in the Scrum methodology.
- At the other end of the scale, SE practices with the lowest usage were mostly diagramming and documentation techniques associated with high ceremony development approaches. Given that Scrum is a low ceremony, agile methodology; this finding is supported by the literature.

- As would be expected, SE practices with the highest usage in the traditional approach were the functional, non-functional and design specifications together with use case narratives (all high ceremony, document driven practices).

Evaluating the usage of engineering practices

- The practice providing the greatest difference in usage between the Scrum and the traditional approaches was the retrospective/ review practice (Scrum 100% usage, Traditional 52% usage, 48% difference). While both approaches allow the development team to reflect on their progress and performance and so improve their ability to deliver quality software, the iterative approach provides a much richer environment to inspect and adapt. This finding is therefore in line with the literature (Cockburn 2001).
- In summary, although Scrum does not prescribe the use of SE practices, they are used extensively. As expected, they reflect the low ceremony, iterative nature of the Scrum environment.

Importance of software engineering practices

The previous section showed clearly that there is a wide spread of SE practices in organizations using the Scrum methodology. This section focuses on the level of importance given to each of these practices in the software development process.

The authors began by analyzing the importance of the various software engineering practices in each phase of

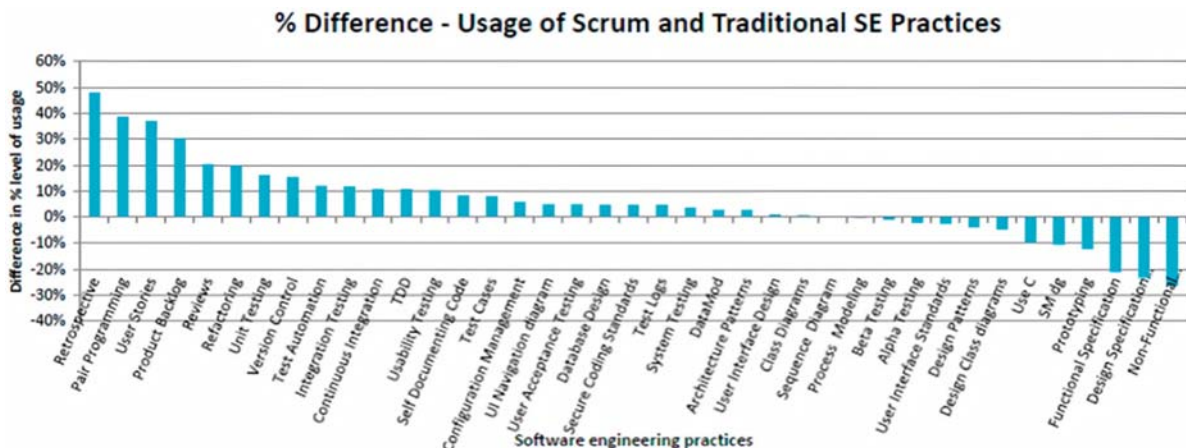


Figure 8: Difference in usage of SE practices in Scrum and traditional method.

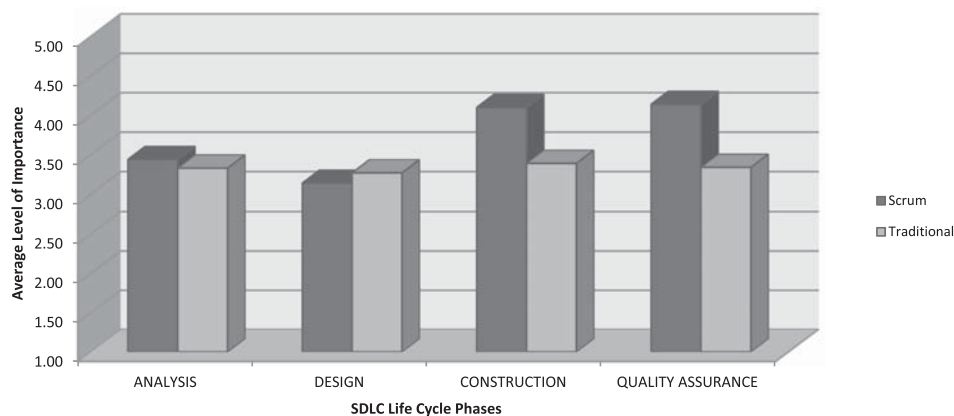


Figure 9: Importance of software engineering practices by life cycle phase.

the software development lifecycle. Means for the level of importance (LOI) of each SE practice in the traditional methodology and Scrum environments were calculated and averaged to obtain an overall mean for each phase in the SDLC. Figure 9 illustrates the following:

- **Analysis Phase:** The literature suggests that by adopting agile methodologies like Scrum, software development teams are likely to reduce the level of ceremony, especially in the analysis phase of the development process. Against such expectations, Figure 11 illustrates the fact that Scrum development teams still consider most tools and techniques in this phase to be fairly important with an average level of importance (LOI) of 3.44, This is almost the same level of importance as in the traditional approach (3.33). Detailed analysis of the various practices within the Analysis phase may help to explain this observation and are dealt with later in the paper.
- **Design Phase:** The average level of importance of design practices appears to agree with the literature, in that most of the design practices used in the traditional methods are still relevant in Scrum. Hence, LOI of 3.14 compared to 3.27 for the traditional method.
- **Construction Phase** Figure 9 shows that Scrum development teams rate these practices high in importance (4.1), even more important than in the traditional setting which is rated 3.39. Given the increased complexity of coding and testing in an iterative environment, the finding is appropriate.
- **Quality Assurance:** Respondents consider the QA practices in Scrum to be very important, given its LOI of 4.11. This was again expected from the literature, given that quality is heavily emphasized in agile development environments (Pudusserry 2009).

To summarize, the analysis of software engineering practices by life cycle phase highlights two important aspects:

- *Firstly*, the overall average level of importance of these practices in Scrum is 3.70 and 3.33 for the traditional methodology.
- *Secondly*, their importance seems to be more focused on the Construction and Quality Assurance phases of the

life cycle rather than on Analysis and Design. This finding was expected as the ‘build’ phase in an agile, iterative environment is more challenging than the traditional waterfall approach and requires rigorous coding and testing practices.

Analysis phase: In a more detailed investigation of the importance of specific SE practices in the analysis phase of the life cycle (see Figure 10), the following was observed:

- As expected, the popular agile requirement elicitation and specification practices are rated as the most important in Scrum. These included User stories (LOI score of 4.5) and the Product Backlog (4.35).
- In addition, more traditional analysis practices including the Functional Specification (LOI 4.0), Process modelling (LOI 3.5), and Use cases (LOI 3.3) were less important in Scrum (3.13, 3.11, and 3.0 respectively).
- Also as expected from the literature, the Functional Specification (FS) document was identified as the most important practice in the traditional approach. However, the Scrum methodology is a low ceremony approach where the formal FS is often replaced by User stories and face to face communication. A low level of importance was expected and yet this practice was rated (LOI 3.13) in Scrum. This finding will be investigated further.
- The balance of the Engineering tools and techniques such as Data modelling, Non-functional specifications, and Prototyping are considered equally important in both Scrum and traditional software development settings. This finding is also expected as the techniques are important development practices and should be followed in all SD approaches.
- One anomaly in the chart of analysis practices is the low level of importance given to Class Diagrams in the Scrum methodology (LOI 2.65) as opposed to the traditional approach (LOI 3.40). Possible explanations for this could lie in the fact that in more recent agile and iterative development literature, the Analysis Class Diagram is referred to as a Concept Diagram (Larman 2004), or Domain diagram (Evans 2004).

Design phase: Following on from the analysis phase, a more detailed review of the importance of specific SE

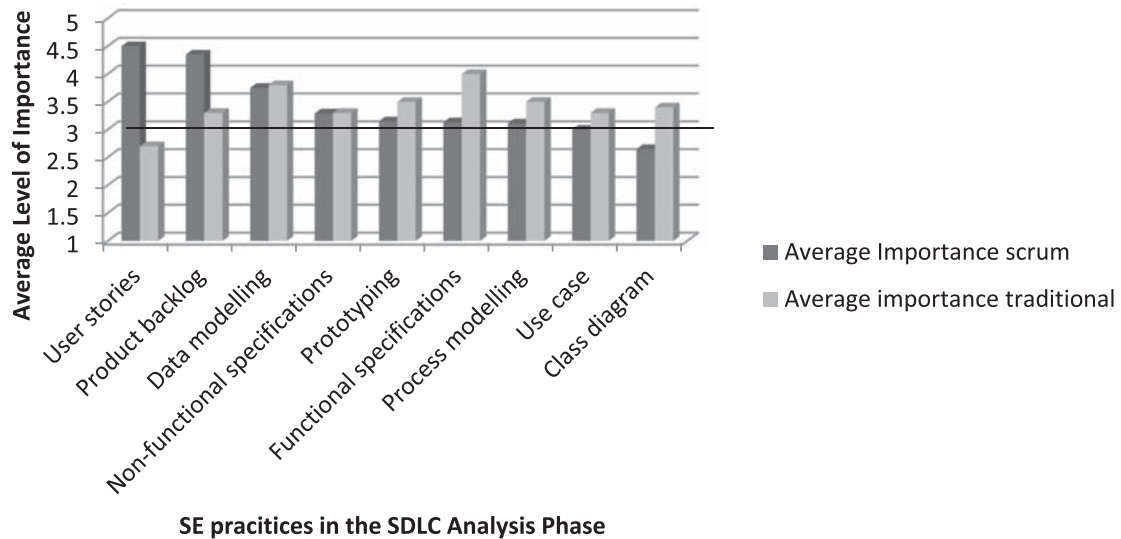


Figure 10: The analysis phase – average level of importance of software engineering practices (sequenced by Scrum level of importance).

practices in the design phase of the life cycle (see Figure 11) was completed:

- Given that design is a critical aspect of software development; the literature believes that design practices are as important in Scrum as they are in the traditional methodologies (Capitalhead 2006; Pudusserry 2009). In addition, none of the design practices identified are uniquely related to the traditional or agile approaches. Figure 11 illustrates little difference in the importance of the design tools and techniques between the two approaches with the majority of practices rated slightly higher in the traditional approach. For instance, Database design (LOI 3.9 in the traditional is 3.69 in Scrum), Architectural patterns (LOI 3.8 in the traditional is ranked 3.58 in Scrum), and Design patterns (LOI 3.7 in traditional is 3.44 in Scrum). The design practices of least importance to Scrum were all diagramming practices and this can be expected given Scrum’s ‘just enough’, low ceremony approach to documentation.
- One other interesting observation is the importance level given to the Design specification, LOI 3.3 in traditional

and 3.15 in Scrum. Like the functional specification, both documents are the backbone of the traditional waterfall approach and were expected to be very unimportant in Scrum. This finding is discussed further in the next section.

Construction phase: As shown in Figure 12, a detailed review of the SE practices in the Construction phase highlighted the following:

- All the practices listed in this phase have average level of importance well above 3.0. This implies that they are considered relevant and important in software development irrespective of which methodology is being used.
- Each of the practices in the construction phase is considered to be more important in Scrum than in the traditional methodologies.
- The literature stresses the significance of managing and controlling coding and testing in an iterative environment (Balsamo et al. 2004), and this is supported by the 10 highest ranking Scrum Construction engineering practices, all with a LOI of above 4. This list includes

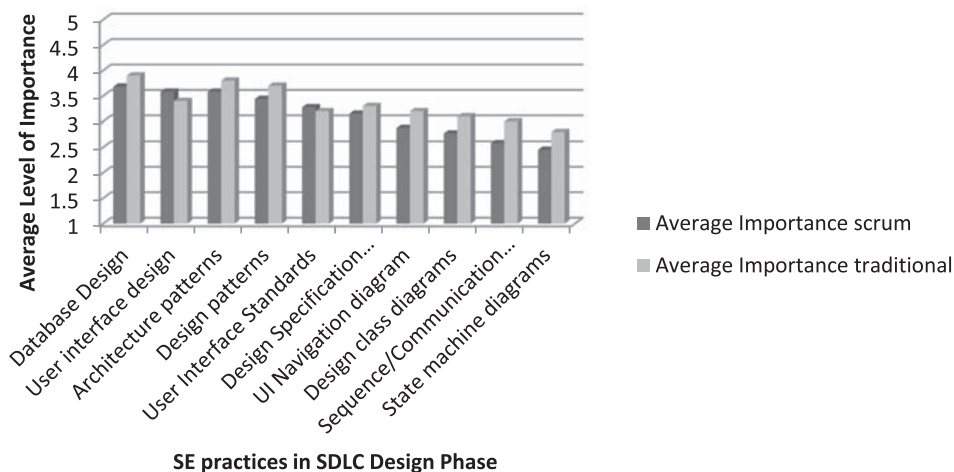


Figure 11: The design phase – average level of importance of software engineering practices (sequenced by Scrum level of importance).

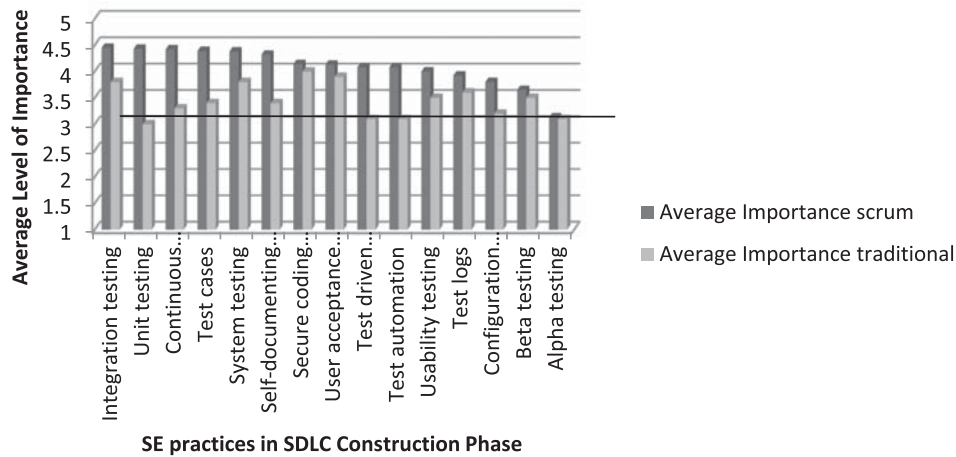


Figure 12: The construction phase – average level of importance of software engineering practices (sequenced by Scrum level of importance).

Integration testing (4.45), Unit testing (4.43), Continuous integration (4.42), Systems Testing (4.38), Test Cases (4.39), Self-documenting code (4.32), Secure coding standards (4.14), User acceptance testing (4.13), Test driven development (4.08) and Test automation (4.08). As explained by one of the Scrum experts interviewed during the study:

Although Scrum does not mandate any engineering practices ... by the nature of the small iterative approaches, it actually demands higher level engineering practices than required in the traditional approach, because without doing things like proper continuous integration, fairly robust testing, version control and those kinds of things, it will be difficult to deliver a quality product

Quality assurance phase: Finally, practices followed in the Quality Assurance (QA) phase of the lifecycle (see Figure 13) were reviewed:

- Similar to the pattern observed in the construction phase, all the engineering tools and techniques listed are considered important (with their LOIs' well above 3.0).

- Also of interest is the fact that each of the practices is ranked as relatively more important in Scrum than the traditional approach, including even the tools and techniques (such as Version Control, and Walkthroughs) that originated in the traditional environment.
- The large gap between the importance ratings of the Retrospective practice in Scrum as opposed to the traditional environment is to be expected as this practice is an integral part of the iterative and agile approach. Moreover reflection (project reviews) is usually completed once at the end of a waterfall project as opposed to the frequent retrospective sessions held in Scrum. However, it is interesting to observe that the iterative practice of Refactoring is ranked important (LOI 3.7) in the traditional environment.

Based on the average level of importance of the various technical practices in the SDLC stages, the respondents consider the practices listed in Table 3 as the most important (LOI above 4.0) software engineering tools and techniques while Table 4 shows the least important ones (LOI below 3.0). It is essential to note that the

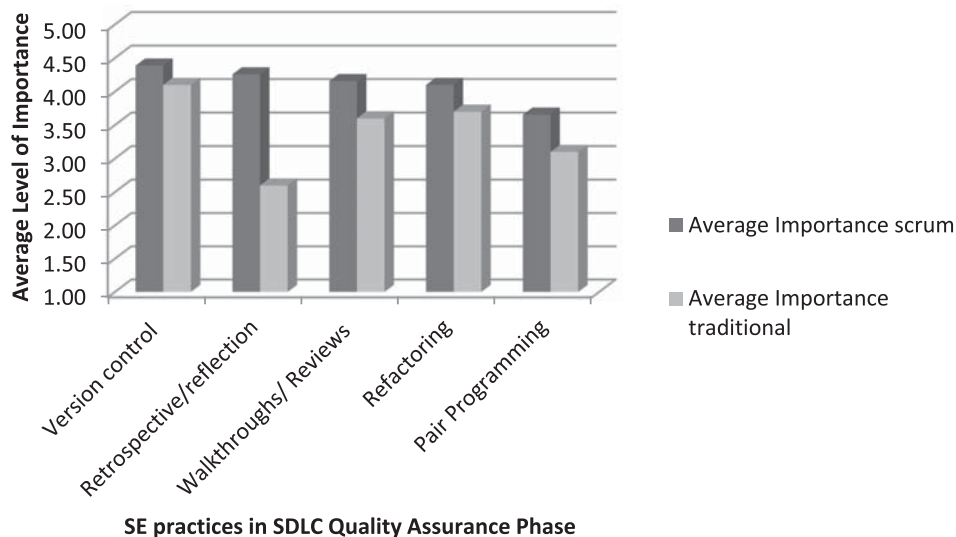


Figure 13: The quality assurance phase – average level of importance (sequenced by Scrum level of importance).

Table 4: The least important software engineering tools and techniques used in Scrum.

Ranking	Non-important engineering practices	Average level of importance (LOI)	SDLC phases
1	State machine diagrams	2.45	Design
2	Sequence/Communication diag.	2.58	Design
3	Class diagram	2.65	Analysis
4	Design class diagrams	2.77	Design
5	UI navigation diagram	2.88	Design

majority of the most important SE tools and techniques belong to the Construction and QA phases of the SDLC, while 80% of the least important tools and techniques belong to the Design phase. This observation shows that an iterative software development approach, such as Scrum, considers the practices in the Construction phase and Quality Assurance to be the most important and those in the Design phase to be the least important, in the development process. It is also interesting to see two low ceremony but critical agile analysis practices rank first (1st) and eighth (8th) in Table 3 as expected from the literature (Scaffidi and Shaw 2007; Pudusserry 2009). Further, as predicted by Fowler (2005), 80% of the QA practices are also regarded as important (ranked 4th, 10th, 11th and 14th in Table 3).

Most of the SE practices with LOI below 3.0 are diagramming techniques (as shown in Table 4). Given the low ceremony approach of agile, this finding is supported by Balsamo et al. (2004) in the literature.

Given the low sample size ($n = 32$) and the variation in respondent roles, a difference of means test was conducted to determine the significance of differences between the importance of practices in the Scrum and Traditional development environments. The test showed p values of less than .05 for a number of practices (see Table 5) indicating the level of importance of these practices was significantly different between the two development environments. The tests show that the practices that are specifically Scrum/agile/iterative practices or are high ceremony tend to have significant differences (p -value of

less than .05). This was expected and supported by the literature.

A set of the t-test was conducted on the importance of software engineering practices in Scrum and Traditional. At p -level 0.05, the result was statistically insignificant at the phase level. This shows that at the phase level software engineering practices are important both in Traditional and Scrum. Table 5 shows the means at the phase level and the p -value. This confirms the findings of the literature regarding the importance of SE practices in Scrum as well as in Traditional.

Qualitative results

Interviews

The interview schedule attempts to identify what practices are currently being adopted by Scrum teams. The purpose of the interviews goes beyond identifying Scrum SE practices and tries to understand how Scrum teams identify and adopt SE practices.

Question 1: Given there are no prescribed software engineering (SE) within Scrum, how do your teams adopt appropriate practices?

The responses to this question can be categorized into three cases:

In the **first case**, the development team with the support and trust of top management are allowed to use any engineering practices whatsoever that work for them as long as they deliver on time and within budget. In this scenario, most teams continue using the engineering tools and techniques brought along from the old traditional development environment but allow flexibility

Table 5: Difference of means test (T-Test) – Significance of level of importance test.

Practice	Approach	Mean	Std Dev	P Value
Functional specification document	Traditional	3.92857	1.33135	0.03802
	Scrum	3.23077	1.06987	
User Stories	Traditional	2.73684	1.32674	0.00004
	Scrum	4.41176	1.01854	
Product Backlog	Traditional	3.16667	1.33945	0.00641
	Scrum	4.27273	1.20605	
Self-documenting code	Traditional	3.50000	1.27321	0.03111
	Scrum	4.34783	0.90738	
Continuous integration	Traditional	3.36842	1.11056	0.00159
	Scrum	4.38462	0.95119	
Test cases	Traditional	3.53846	1.36325	0.02566
	Scrum	4.30303	1.13150	
Test driven development	Traditional	3.10000	1.16701	0.00729
	Scrum	4.03704	0.94000	
Test automation	Traditional	3.11765	1.20428	0.02226
	Scrum	4.11538	0.95192	
Unit Testing	Traditional	3.04762	1.34693	0.00072
	Scrum	4.36667	0.87471	
Retrospective/reflection	Traditional	2.46154	1.34164	0.00056
	Scrum	4.23529	0.81868	

in their use in that they keep the practices that work for them and discontinue those that do not.

One of the ways they adopt new engineering practices is through the retrospective, which is where they ask themselves how to solve problems they seem to be facing ... 'and they say well we should try this thing'(D1).

Scrum is **inspecting** and **adopting**, and it allows us to have these discussions in retrospectives. (SM2)

In the **second case**, the development teams are mandated to use a specified set of tools and techniques by the management.

I am a 'benign dictator', and I absolutely believe being prescriptive. I impose the engineering practices and the team have to adhere to them. What really happens is that they gauge what they can do based on what requirements I put in place; for instance, I require that we have unit test, TDD, and write test cases at certain level, and I also require that we don't write new software until we fix all the bugs ... these are just part of the engineering practices. (M1)

So, until such time that you have a Scrum which is matured and can make decisions based on knowledge of the Scrum process ... and do it by the book (have your stand-up meetings, task board, retrospective and so on), because you need to first understand the mechanics. (M1)

This expert believes that engineering practices must be imposed on the development team right from the beginning. The expert says that 'in engineering practice one needs to be very steadfast, and be very disciplined', for instance, considering the fact the team has a short time (in the case of this expert, a sprint period of fifteen working days) to deliver a product. There must be some prescriptions such as the level of test code, regression testing and code review required in the development process.

In the **third case**, however; some experts argue that it should be a mix of the two cases discussed above. For instance:

There are two aspects: there is a role at the organizational level that can be played, but largely Scrum promotes self-organization, and self-reliance; whereas at the management level, many levels away from the software development engine, they should trust the development team to do what is right and it's the responsibility of the development team as professional software developers to do what is necessary to get the job well done. Management's role is really more of a guidance and not prescriptive. (C1)

Another expert says:

I think it's quite okay for the company to impose some practices, because we don't want to start from the first principle every time; however, the teams should also be given opportunity to explore and adopt new engineering practices **based on what causes them pain** ... I don't think that we should pretend as if we have not done software development project before ... [*sic*] rather we should make use of some of the knowledge and experience we have from the previous projects. So, we [management] try not to be too prescriptive, but also by default don't want to reinvent the wheel each time. For instance, without having some robust testing and without having good sort of version control, you just don't have the time to cope without them. (M3)

In the last statement, the respondent suggests that the teams adopt practices that offer answers to their unanswered questions. The following quote summarizes:

There are things that we, as the company mandate, then the teams will often extend that [the engineering practices] based on where they, in retrospective, come up with **pain points**. (M3)

In the words of another respondent:

Generally, what happens is that people [development teams] start looking for the 'better' engineering practices to adopt, because already something is broken and if the team find themselves lacking on how to implement such techniques(s), that's when the role of a coach or someone with experience can **nudge** them along and **push in the right direction**. (C1)

Having said that, the assistance of competent Coaches, or Scrum Masters are often advised as this helps to accelerate the teams' productivity and learning process. As mentioned by one of the experts:

I can explain Test Driven Development to you in five minutes, but it may take three years to fully understand the practice and its values. (C1)

In view of the above arguments, experts stress that it is very important that the team be **willing** to adopt these practices. While management may tell development teams to start adopting some XP practices, such teams will only adopt the practices in order to keep management happy without much value to the team development.

Question 2: Is there governance at the organizational level over documentation (artefacts and level of ceremony)?

Given that the agile approach values working software over documentation, a vast majority of the respondents claim that they use **just enough** documentation for their development projects.

We need **as much documentation** as is **necessary** to **communicate** clearly what's needed, and that's very much going to depend on the type of project, the complexity of things ... If you have a system with complex business rules, you'll be crazy to not document! (That is, it is mandatory that you have a detailed documentation) It's really what is the level of communication that is needed for the team ... in traditional waterfall approach, the documentation is often there for contractual purposes, and also there because of ... the time between when something (system) was last design and when it's going to be coded, whereas if you're doing just-in-time documentation and just-in-time communication, the level of the documentation can be different. (M3)

Another says that:

It depends on what the needs of the companies are. If you are working in an environment that requires you to have specific documents for legal requirements, then you need detailed documentation. (M4)

In essence, most of the respondents claim that agile gives you the opportunity to build your ceremony to best suit your situation.

Our documentation is not heavy in any way, rather it is sufficient to promote quality software ... My honest belief is that if any practice adds overhead without adding any values they should die lonely death, because

If I cannot justify (to any of my developers) why we are doing something then we shouldn't be doing it. So, our specifications are our stories, and our test cases come out of it. (M1)

Question 3a: Do you think the lack of SE practice can impact on the quality of the developed system?

The majority of the respondents argue that Scrum might be a management methodology, but it does mandate the development team to make use of engineering practices, even though it does not prescribe which to use.

Although Scrum doesn't mandate SE practices, it demands it! (M3)

The expert emphasizes the fact that without engineering practices, there is no way quality can be guaranteed; as such it is critical that the teams adopt various practices in order to succeed.

It was argued that the fact that a team is developing software means that they have engineering practices in existence:

By default, if you write software, you have engineering practices, these practices may not be efficient, but at least they do they exist [*sic*]. (M4)

Question 3b: Do you think the lack of SE practice is a barrier to Scrum adoption?

The lack of software engineering practices in Scrum (according to the respondents) does not cause confusion in the team.

On the contrary, many respondents believe that Scrum helps to expose the main issues in the development process. It does so by facilitating better communication and transparency and in particular, the retrospective meeting helps the team to identify what is missing and how it can be best fixed. As explained by one of the respondents, it is through the retrospective that the team decides which engineering practices are the best for them. Some experts argue that the fact that Scrum does not prescribe any engineering practices is the methodology's strength, since different projects may require different practices; and these practices are often evolved through small iterations and in retrospect.

The thing about Scrum is that it exposes problems really quickly. (SM2; D1)

In summary; in spite the fact that Scrum does not recommend SE practices, the practices do exist nevertheless (although they might be inefficient). By the use of the '*inspect and adapt mechanism*' of Scrum, the development team can ensure that the quality of their products is high. With this understanding in a team, the lack of prescribed SE practices in Scrum does not constitute a barrier to its adoption.

Question 4: Do you think the combination of Scrum and XP is an appropriate approach to take?

Some of the interviewees explained that the Scrum teams must focus on any technical skills that help them solve the particular problems they are experiencing, even if it is a mixture of practices across different methodologies.

We adopted a hybrid system (Scrum and XP) to cater for the problem that we had. I think every organization faces

different difficulties and should adopt the appropriate engineering practices based on their needs. (M1)

In support of M1's explanation,

Scrum and XP is pretty normal and XP in terms of its engineering practices were seen as the ideal set of practices for Scrum. Also given that XP is more of an engineering framework whereas Scrum is seen as Project Management framework, the two will really work well together ... FDD can also exist very well in the context of Scrum. (M3)

Respondents such as M4 suggest that a Scrum team has to learn to use any practices that will help them build their desired product. For instance, if the team has XP techniques in place already, then the members have to learn how to use them, but if the team has never implemented XP before, then the team members have to do inspect and adapt to identify technical practices that will solve their problems. However, as the team keeps on inspecting and adapting their development process, they might throw away some practices that do not work for them and pick up others which do.

Then there is an argument that they will maybe eventually invent XP, because the team might say what is the problem ... ok we need to do code review, or we need to share knowledge and ownership, or let's do continuous integration ... you build these practices in because they solve your problem although they might not be exactly the same as XP, but you still pick up those ideas relating to XP's ... (M4)

That said, C1 advised that it is better

to adopt the engineering tools and techniques a little bit at a time as opposed to the whole framework or management methodology. (C1)

As mentioned in the literature, Scrum's and XP's practices can complement each other (Schwaber and Beedle 2002). The appropriate application of the 'inspect and adapt' of Scrum will enable a team to properly analyze problems and identify the engineering tools and techniques that will solve them, and they will also be able to decide how much of any technique to adopt at a given time.

Discussion and implication

As already outlined in the literature; Scrum is a lightweight project management approach to software development. The first important finding of the study was that while Scrum does not prescribe SE practices, these SE practices are extensively used and rated as more important in Scrum than in the traditional software development methods. As one of the interviewees said, '*if a team writes software, then the team has engineering practices, the practices may be a mess, but they do they exist nonetheless.*' Many Scrum teams have experienced success by pairing the management practices (of Scrum) with engineering practices from other methodologies (such as XP). Surprisingly, a majority of the respondents in the study consider the engineering tools and techniques to more important in Scrum than in the traditional environment. The overall average level of importance of various technical practices in Scrum was found to be 3.70 while that of the traditional methodologies was

3.33; this reflects how important these practices are in any software development process. In another relevant observation (see Figure 11), the various engineering tools and techniques in the construction and quality assurance phases of the SDLC are considered to be more important than those in the analysis and design phases of the life cycle. This outcome is in line with the literature's view that iterative software development processes such as Scrum place emphasis on Construction and QA.

The subject of agile methodologies (such as Scrum) and software documentation continues to be controversial. This may be attributed to the different interpretations given to the following value of the agile development, '*working software over comprehensive documentation*'.

This manifesto value is one of the fundamental criteria often used to distinguish between an agile process and a traditional software development approach (Highsmith and Cockburn 2007). As explained in the literature review section of this study, authors such as Larman (2004) and Kruchten and Kroll (2003) equate agility with lightness in documentation; and as such, less importance being assigned to the practice of documentation in Scrum. This suggests that less importance would be assigned to documents such as the functional specification. However, the result of the questionnaire coupled with the comments of the experts suggest that documentation is considered valuable in the Scrum environment. For instance, the LOI of the functional specification, and design specification were expected to be below the 3.0 threshold. Detailed explanations were given by the respondents to support this 'anomaly'. According to most of the experts, the above agile value does not necessarily translate into fewer documents or lack of documentation. The manifesto value simply means that working software should be the team's goal, and if producing detailed documentation will inhibit the development process, then it should be adjusted. This is why, one of the interviewees said:

If anything adds overhead without adding value it should die a lonely death.

Does the lack of engineering practices serve as a barrier to scrum adoption? The literature was clear on this issue.

When teams adopt the Scrum process, they go faster, show progress, things look good ... and then the quality becomes a problem. Now the team are fighting through quicksand ... the code quality is poor and developers are expected to continue to make progress ... the team is heading for burn-out. (Kelly 2010)

This is what happens when a software development team adopts Scrum without software engineering practices such as Test Driven Development, continuous integration and refactoring (Pudusserry 2009; Allan Kelly Associates 2010). In view of Allan Kelly Associates' (2010) comment, the lack of engineering practices in Scrum can constitute a barrier to a successful implementation of the Scrum practices. However, according to one of the interviewees; although Scrum may lack engineering practices (this is what Schwaber (2010) refers to as 'holes'), Scrum does make provision for finding

methods to fill in these holes: it's the '*inspect and adapt*' principle. According to the respondent:

In planning phase, the team inspect and adapt their planning process; during code review the Scrum team is able to inspect and adapt their software products; in retrospective, the team inspect and adapt their development process

Another interviewee claimed that Scrum helps to expose problems really quickly and by the use of the '*inspect and adapt*' approach, team communication will improve, and this will eventually lead to better software quality.

That said, in the words of Cohn (2011):

The fact that Scrum is silent on many things (such as which engineering practices to adopt) is both its strength and weakness. Scrum, for example, does not say a team should use version control system. Yet, all the good Scrum teams I've seen do". A good Scrum team will find appropriate technical practices to ensure high quality. A bad Scrum (if they can be said to be doing Scrum) will use Scrum as an excuse to do low quality work

If the options are either to adopt Scrum with a set of prescribed SE practices or to find these practices overtime; the latter approach seemed to be favoured by all the Scrum Masters interviewed. Given this strategy, the lack of SE practices was not seen as a barrier to adoption of Scrum. In view of the above arguments, it can be concluded that the lack of engineering practices in Scrum is not a barrier to the adoption process, so long as the team follow the Scrum's basic philosophy of inspect and adapt.

Software development depends considerably on team performance, as does any process that involves human interaction (Moe, Dingsøyr, and Dybå 2009). As seen in both the literature and interviews, software development teams adopt relevant engineering practices in any of the following ways:

- (i) Command-and-control approach
- (ii) Self-Managed Team approach
- (iii) Combination of command-and-control and Self-Managed Team approaches

The command-and-control approach follows the traditional perspective on software development, which promotes a plan driven product-line strategy to software development using a standardized, controllable and compulsory software engineering process (Moe, Dinsøyr, and Dyba 2010). The use of a command- and-control approach in a Scrum environment implies that the management will impose engineering practices on the team; the team has no say in the selection of technical tools nor about which techniques are to be implemented. Two of the respondents agreed with the implementation of the command-and-control approach. According to M1, the Scrum teams require a certain amount of time to properly understand and become familiar with the development process as well as the engineering practices being used. Given that there are industry-best practices for software engineering tools and techniques, as well as that management cannot afford to wait while the

development team ‘experiments’ with engineering practices, it is considered normal to impose ‘best’ practices on the team.

In contrast, the rest of the respondents argue that teams perform better, and value their practices, if they are given enough time to find and fit the software engineering practices that match their development style, as opposed to being forced to use particular ones. This may be referred to as the ‘Self-Managed Team’. According to Highsmith and Cockburn (2007), a self-organizing team is able to make decisions themselves and continuously adapt to changing situations. According to one of the interviewees, SM2, such teams effectively make use of the ‘*inspect and adapt*’ mechanism of Scrum. As these teams inspect and adapt different situations and learn new skills such as TDD, and Continuous Integration, they pass through various stages of learning as described in the literature review. The Shu-Ha-Ri model, Dreyfus model and the Road to Mastery were discussed. Interviewees also highlighted the principles underpinning the Shu-Ha-Ri and following the Road to Mastery methods. For instance, using the Shu-Ha-Ri model, according to Cockburn (2001) in the phase of Shu (following), the team follows the teachings of a coach or any other specialist precisely. At this stage, the team concentrates on how to complete their tasks (such as regression testing and continuous integration), without worrying too much about the underlying theory. If there are multiple variations on how to solve a problem, the team focuses on just the one procedure the coach teaches them. For instance, the team may only focus on using continuous integration (Cockburn 2001; Kim 2007).

In the phase of Ha (detaching), the Scrum team appreciates the limitations of the single procedure and looks for rules about when the procedure breaks down. The team also learn to adapt the single procedure to various cases. The team is now willing to learn the other procedures in an attempt to discover which procedures are most applicable and the circumstances that make each procedure break down (Cockburn 2001; Kim 2006). In the phase of Ri (fluent), the team’s knowledge has become integrated through a thousand thoughts and actions. It can now improvise around one procedure or make up new procedures because the team members now understand the desired end result and simply make their way towards that end (Cockburn 2001; Kim 2006). To summarize; at the end of Shu, what the team sees is nothing but the rules – everything looks like the rules. At the end of Ha, what the team sees is nothing like the rules. At the end of Ri, the team apply their minds (Kim 2006). Nevertheless, many of the respondents believed that an appropriate combination of the command-and-control and Self-Managed Team approaches will work best. They argued that in order to explore the benefits of both methods, the management should mandate a base set of practices, while at the same time allowing the team to augment them with technical practices of their (team’s) choice.

The most commonly used agile methodologies are Scrum and XP (Digital.ai Software Inc. 2020). Scrum (37%), Scrum/XP hybrid (23%), and XP (12%) together represent almost 80% of agile software development. The interviewees all agreed that Scrum is a management

practice and XP is focused on engineering practices. As explained by respondent C1, the use of Scrum will help a development team enhance communication in the team and develop many other soft skills. According to Singh (2009, 8), ‘Both XP and Scrum promote better collaboration, communication, iterative/incremental development and frequent releases. Or in short take baby steps!!! [*sic*]’ However, these achievements derived from the use of Scrum will not necessarily translate into quality software, unless the technical practices are fine-tuned. According SM2 (one of the interviewees), Scrum provides the platform for the team to get together (via retrospective sessions) and discuss how they can improve and solve the problems that the team is facing. As explained by another respondent, M3; by inspecting and adapting, Scrum teams are able to select the XP practices that suit them the most or evolve their own technical practices; which may come from any existing methodology such as XP, FDD, Lean or other methodologies. This is why Cohn (2010, 1) usually says to Scrum teams ‘start with Scrum and then invent your own version of XP’. Having said that, almost all the interviewees support the idea that the adoption of engineering practices alongside Scrum practices, must be done bit by bit, given that engineering techniques are relatively harder to implement than Scrum practices. Pudusserry (2009) also agrees with the idea of incremental adoption of the relevant engineering practices.

Conclusion

The objectives of this research were to identify the importance of software engineering practices in the Scrum development environment, and how Scrum teams adopt an appropriate set of SE tools and techniques. Further, the study sought to understand whether a Scrum team should start Scrum alongside all the engineering practices of XP or whether the practices should be adopted incrementally. This research was conducted by examining the sample data from various industries such as communications and embedded systems, financial asset management, software development houses and consulting. The study had both quantitative and qualitative aspects. Regardless of the fact that Scrum does not explicitly recommend engineering practices, the respondents ranked the engineering practices as more important in Scrum than in the traditional methodologies. Another significant observation from the study was that the various engineering tools and techniques in the construction and quality assurance phases of the SDLC were considered as more important than those in the analysis and design phases of the life cycle. The study also found that the lack of software engineering practices does not constitute a barrier to a successful adoption of Scrum, provided the ‘*inspect and adapt*’ principle is properly followed. That is, as the Scrum team *inspect and adapt* their planning stage, the planning process will be improved, as the team *inspect and adapt* code review, better quality software will be produced and as the Scrum team *inspect and adapt* the development process, SE practices will be made more efficient. Regarding how the Scrum teams adopt the appropriate set of SE tools and techniques, the paper identified three possibilities.


The first is referred to as the *command-and-control approach*. In such a case, the management of the business imposes the engineering practices, to which the Scrum teams must adhere. The second possibility is called the *Self-Managed Team approach*. This is where the Scrum team is left to experiment with various engineering tools and techniques, and eventually come up with the practices that best suit them. This approach is considered to be more favourable to the team than the top down one, given the fact that team members in the former environment will grow to better understand and appreciate the practices as they follow the various learning models such as Shu-Ha-Ri, Dreyfus and Road to Mastery. The third possibility is simply the combination of the command-and-control and the Self-Managed Team approaches. In this situation, certain engineering practices are mandated by the management while the Scrum teams are given freedom to evolve additional technical practices that suit them. With that said, from the interviews it was apparent that the majority of the organizations implement the Self-Managed Team approach. Nevertheless, in any of these possibilities, the assistance of coaches, Scrum Masters, consultants and other specialists may be requested to help the teams. Given that it is easier to adopt the management-oriented Scrum practices than the XP technical practices (Puduserry 2009), and based on the data analysis and findings, this research found that it is often better for a development team to start with Scrum, and then later introduce the suitable technical practices of XP to the team.

The outcomes of this empirical research will give guidance to software development teams that have recently adopted (or are about to adopt) Scrum in their selection of the appropriate engineering practices. It will also raise the awareness of managers regarding the three ways in which SE practices can be adopted in an organization, that is: the top down, bottom up or a combination of the two. Furthermore, it will educate and raise the awareness of development teams and decision-makers regarding the various learning processes that development teams follow to adopt new practices in Scrum. Finally, it will give guidance to both organizations and the teams on whether the Scrum teams should adopt XP engineering practices incrementally or follow a 'big bang' approach. As regards limitation of the study, there are respondents that ranked some unused SE practices, making it difficult to justify the integrity and validity of the average / percentage usage of each SE practice. Thus, the calculation of average / percentage usage was discouraged. Given the limitations in scope and time, it was not possible to address all key areas outside, but related to, the main focus of this research. The research team recommend that future empirical studies should be conducted on how a development team that has recently migrated to Scrum progress along the various maturity models such as the Shu-Ha-Ri, Road to Mastery and Dreyfus.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID iD

Ismaila Temitayo Sanusi  <http://orcid.org/0000-0002-5705-6684>

References

- Abrahamsson, P., and O. Salo. 2008. "Use and Usefulness of Extreme Programming and Scrum." *IEEE Xplore Journal* 2 (1): 1–7.
- Allan Kelly Associates. 2010. "The Scrum Wall (Another Agile Failure Mode)." Accessed 20 September 2020. <https://www.allankellyassociates.co.uk/archives/869/scrum-wall-another-agile-failure-mode/>.
- Appelo, J. 2008. *Management 3.0, Leading Agile Developers, Developing Agile Leaders*. Berlin: Springer Verlag Publishers.
- Awad, M. A. 2005. *A Comparison Between Agile and Traditional Software Development Methodologies*. Western Australia: School of Computer Science and Software Engineering, The University of Western Australia.
- Balsamo, S., A. Di Marco, P. Inverardi, and M. Simeoni. 2004. "Model-Based Performance Prediction in Software Development: A Survey." *IEEE Transactions on Software Engineering* 30 (5): 295–310.
- Bang, T. J. 2007. "An Agile Approach to Requirement Specification." *8th International Conference on Agile Processes in Software Engineering and Extreme Programming*, 193–197. Berlin: Springer-Verlag.
- Bjeirmi, B. F., and A. K. Munns. 1996. "The Role of Project Management in Achieving Project Success." *International Journal of Project Management* 14 (2): 81–87.
- Braun, V., and V. Clarke. 2006. "Using Thematic Analysis in Psychology." *Qualitative Research in Psychology* 3 (2): 77–101.
- Brhel, M., H. Meth, A. Maedche, and K. Werder. 2015. "Exploring Principles of User-Centered Agile Software Development: A Literature Review." *Information and Software Technology* 61: 163–181.
- Cao, L., and B. Ramesh. 2008. "Agile Requirements Engineering Practices: An Empirical Study." *IEEE Software* 25 (1): 60–67.
- Capitalhead. 2006. "Software Development." (June 11). Accessed August 12, 2011. <http://capitalhead.com/solutions/software-development.aspx>.
- Cho, J. 2009. "A Hybrid Software Development Method for Large Scale Projects: Rational Unified Process with Scrum." *Issues in Information Systems Journal* 10 (2): 338–347.
- Cockburn, A. 2001. *Agile Software Development*. Boston: Addison-Wesley Professional.
- Cohn, M. 2007. "Differences Between Scrum and Extreme Programming." Accessed August 11, 2011. Mountain Goat Software: <http://blog.mountaingoatsoftware.com/differences-between-scrum-and-extreme-programming/comment-page-1>.
- Digital.ai Software, Inc. 2020. "14th Annual State of Agile Report." Texas, United States: Digital.ai Software, Inc. <https://digital.ai/catalyst-blog/the-14th-annual-state-of-agile-report>.
- Drnovscek, S., and V. Mahnic. 2005. *Agile Software Project Management with Scrum*. Ljubljana: University of Ljubljana, Faculty of Computer and Information Science.
- Dyba, T., and T. Dingsoyr. 2008. "Empirical Studies of Agile Software Development: A Systematic Review." *Information and Software Technology* 50 (9-10): 833–859.
- Eckstein, J. 2013. *Agile Software Development with Distributed Teams: Staying Agile in a Global World*. Boston: Addison-Wesley.
- Etikan, Ilker, Sulaiman Abubakar Musa, and Rukayya Sunusi Alkassim. 2016. "Comparison of Convenience Sampling and Purposive Sampling." *American Journal of Theoretical and Applied Statistics* 5 (1): 1–4. doi:10.11648/j.ajtas.20160501.11.

- Evans, E. 2004. *Domain-Driven Design Tackling Complexity in the Heart of Software*. New York: Addison-Wesley Publishers.
- FreeTutes.com. 2011. "Waterfall Software Development Life Cycle Model." June 26. Accessed at <http://www.freetutes.com/systemanalysis/sa2-waterfall-software-life-cycle.html>.
- Fowler, M. 2005. *The New Methodology*, December 13. Accessed April 11, 2011. <http://www.martinfowler.com/articles/newMethodology.html#FromNothingToMonumentalToAgile>.
- Fowler, M., K. Beck, J. Brant, W. Opdyke, and D. Roberts. 2002. *Refactoring: Improving the Design of Existing Code 1999*. Boston: Addison-Wesley.
- Highsmith, J., and A. Cockburn. 2007. "Agile Software Development: The Business of Innovation." *IEEE Xplore Journal* 3 (5): 1–3.
- Highsmith, J., and J. Highsmith. 2009. *Agile Project Management: Creating Innovative Products*. Chicago: Addison-Wesley Publishers.
- Hunt, J. 2006. *Agile Software Construction*. London: Springer - Verlag Publishers.
- Jyothi, V. E., and N. K. Rao. 2011. "Effective Implementation of Agile Practices Ingenious and Organized Theoretical Framework." (*IJACSA International Journal of Advanced Computer Science and Applications* 2 (3): 1–8.
- Kanwal, F., K. Junaid, and M. A. Fahiem. 2010. "A Hybrid Software Architecture Evaluation Method for FDD – An Agile Process Model." *The Institute of Electrical and Electronics Engineers*.
- Kerzner, H. 2009. *Project Management: A Systems Approach to Planning, Scheduling and Controlling*. New Jersey: John Wiley and Sons.
- Keshta, N., and Y. Morgan. 2017. "Comparison Between Traditional Plan-Based and Agile Software Processes According to Team Size & Project Domain (A Systematic Literature Review)." In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 567–575. IEEE.
- Kim, Y. 2007. "Analyzing Scrum Agile Software Development with Development Process, Social Factor, and Project Management Lenses." *Americas Conference on Information Systems (AMCIS)*, 1–10. California: Americas Conference on Information Systems.
- Kruchten, P., and P. Kroll. 2003. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Boston: Pearson Education, Inc.
- Larman, C. 2004. *Agile and Iterative Development: A Manager's Guide*. Boston: Pearson Education, Inc.
- Lei, H., F. Ganjeizadeh, P. K. Jayachandran, and P. Ozcan. 2017. "A Statistical Analysis of the Effects of Scrum and Kanban on Software Development Projects." *Robotics and Computer-Integrated Manufacturing* 43: 59–67.
- Lynema, E. 2010. *Agile Project Management and the Real World*. New York City: NCSU Libraries.
- Moe, N., T. Dingsøy, and T. Dybå. 2009. "Overcoming Barriers to Self-Management in Software Teams." *IEEE Software* 26 (6): 20–26.
- Olaleye, S. A., I. T. Sanusi, F. S. Mark, and J. Salo. 2020a. "Customers' Loyalty to Tablet Commerce in Nigeria." *African Journal of Science, Technology, Innovation and Development* 12 (2): 217–229.
- Olaleye, S. A., I. T. Sanusi, and J. Salo. 2020b. "Mobile Customers' Experience and Loyalty: A Study of Tablet Gender Divergence in Finland." *International Journal of Internet Marketing and Advertising* 14 (3): 275–298.
- Podusserry, A. 2009. *Agile Project Management Implementation Approach*. Bangalore: Project Management Research Institute.
- Sahota, M. 2011. *5 Ways Scrum Creates Safety: Why One CSC Uses Scrum and XP Together to Avoid XP Risks*, February 22. Accessed April 8, 2011. Scrum Alliance transforming the world of work: <http://www.scrumalliance.org/articles/180-ways-scrum-creates-safety-why-one-csc-uses-scrum-and-xp-together-to-avoid-xp-risks>.
- Scaffidi, C., and M. Shaw. 2007. "Developing Confidence in Software Through Credentials and Low-Ceremony Evidence." In *Institute for Software Research*, 1–3. Pittsburgh: School of Computer Science Carnegie Mellon University.
- Schwaber K. 2004. *Agile Project Management With Scrum*. Redmond: Microsoft Press.
- Schwaber, K. 2010. "Waterfall, Lean/Kanban, and Scrum." July 3. Accessed August 11, 2011. <http://kenschwaber.wordpress.com/2010/06/10/waterfall-leankanban-and-scrum-2/>.
- Schwaber, K., and M. Beedle. 2002. *Agile Software Development with Scrum*. Upper Saddle River, NJ: Prentice Hall.
- Shore, J., and S. Warden. 2008. *The Art of Agile Development*. California: O'Reilly Media Inc.
- Singh, V. 2009. "Differences Between Scrum and Extreme Programming." Accessed August 11, 2011. Mountain GoatSoftware:<http://blog.mountaingoatsoftware.com/differences-between-scrum-and-extreme-programming/comment-page-1>.
- Srivastava, A., S. Bhardwaj, and S. Saraswat. 2017, May. "SCRUM Model for Agile Methodology." In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 864–869. IEEE.
- Stafford, J., Y. Francino, B. Matturro, and M. Webb. 2011. *Definition Scrum*. Accessed August 12, 2011. SearchSoftwareQuality.com: <http://searchsoftwarequality.techtarget.com/definition/Scrum>.
- Sutherland, J. 2004. "Agile Development: Lessons Learnt from the First Scrum." *Cutter Agile Project Management Advisory Service Journal* 5 (20): 1–6.
- Sutherland, J. 2007. *Deep Lean*. Boston: Massachusetts Institute of Technology publishers.
- Vallon, R., B. J. da Silva Estacio, R. Prikladnicki, and T. Grechenig. 2018. "Systematic Literature Review on Agile Practices in Global Software Development." *Information and Software Technology* 96: 161–180.