A Review of Agile Methods for Requirement Change Management in Web Engineering

1st Abdullah Sabih Alsharari
School of Computer Sciences
Universiti Sains Malaysia
Penang, 11800, Malaysia
upm8@hotmail.com

2nd Wan Mohd Nazmee Wan Zainon
School of Computer Sciences
Universiti Sains Malaysia
Penang, 11800, Malaysia
nazmee@usm.my

3rd Sukumar Letchmunan
School of Computer Sciences
Universiti Sains Malaysia
Penang, 11800, Malaysia
sukumar@usm.my

4th Badiea Abdulkarem Mohammed
College of Computer Science and Engineering
University of Ha'il
Ha'il, 81481, Saudi Arabia
b.alshaibani@uoh.edu.sa

5th Mohammed Sabih Alsharari
Faculty of Computer Science & Information Technology
Universiti Malaya
Kuala Lumpur, 50603, Malaysia
alshararai4m@gmail.com

2023 International Conference on Smart Computing and Application (ICSCA) | 979-8-3503-4705-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSCA57840.2023.10087734

***Abstract*— Web-based systems are essential in many different applications, which makes web-based development different from traditional software development. The agile methodology has gained lots of popularity in the last 15 years and become very traditional and acceptable widely among software engineers; nevertheless, the use of agile methods has extended to cover other areas of software engineering field including Requirement Engineering (RE) that fit the need of web engineering. The pace of the current web's software development is fast and dynamic such that the changes of the requirements during software development and after turning to production phase are possible and recurrent. Therefore, agile software engineering conceptuality has evolved as an adequate approach to overcome changes in the web's software's requirements; due to frequent changes in requirements, web engineers call for help of agile software engineering methods, which strive to truly manage changes in requirements rather than preventing these changes. This paper provides a review on the available agile methodologies that used to assess requirement change management in web engineering.**

Keywords*— Engineering, Agile Methodologies, Requirement Change Management, Software Engineering

I. INTRODUCTION

The use of scientific, engineering, and management ideas and systematic methodologies with the goal of successfully building, implementing, and maintaining high-quality web-based systems and applications, is a description of Web Engineering [1]. Web engineering methods use a variety of notations and recommend some development methodologies, therefore using a standard meta-model as based approaches for the web domain [2]. The modeling models' best delineation is the meta-model. The relationship between meta-models and their features, with aptly-formedness instructions, sustains the need for developing a semantic web [2]. Web engineering methods developed based on these meta-models as part of the meta-models promoted design. The common meta-models should be a combination of the modeling designs of well-known web engineering methodologies, allowing for a thorough review and familiarization [3].

There are various classifications for webs and web apps based on the ongoing expansion of features and complexity. Aghaei et al. [4] categorised four generations based on the evolution of the internet: web 1.0, web 2.0, web 3.0, and web 4.0. The characteristics of the generations are discussed and compared. Since 1989, the web has grown tremendously. It is adopting artificial intelligence approaches more and more, and in the near future, it is expected to become a massive web of sophisticated intelligent communications [4]. There are several levels of complexity in web apps. Depending on their progress history and level of difficulty, they may be only informative or manage full-size and fully-featured 24/7 e-commerce systems [5]. Wakil et al. [6] came to the conclusion that the most recent online apps are omnipresent web apps, intelligent web apps and semantic web apps. Furthermore, there are Rich Internet Application (RIA), which concentrates on the customer and the user interface of the server.

This paper is organized as follow: The following section reviewed the agile methodologies. Section III reviewed the agile development methods. Section IV highlighted the advantages and disadvantages of agile methodologies. Section V reviewed the agile requirement engineering. Finally, Section VI conclude the paper.

II. AGILE METHODOLOGIES

Agile identifies issues that have resulted in software development [7]. The waterfall model's development methodologies have always resulted in the same problem for the last 50 years. The unsuitable approach has always been utilized to specify requirements. Requirements, at their best, can reveal what is desired by the end users. In general, criteria have been developed depending on the end users' needs, who will be unable to do anything with the finished product. What is desired versus what is required are two distinct issues. It is certainly worthwhile to investigate what end users require, rather than simply listening to what they desire.

A. Agile Manifesto

Agile development is constantly changing, where the most important parts of the software development have been split to four values, known as the agile manifesto. Picture of Agile manifesto is shown in Fig. 1.

The following concepts are considered in Agile Manifesto:

1) Processes and instruments are valued less than people and engagement.
2) Working application is valued more than overall documentation.
3) Cooperation with customers is valued more than contract negotiations.
4) Following the plan is valued less than reacting to change.

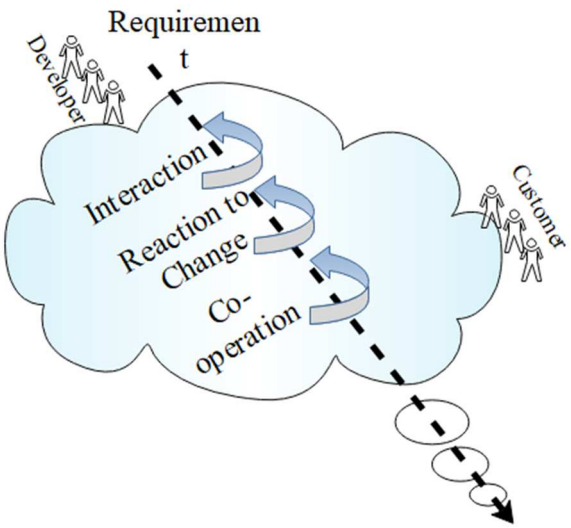

Fig. 1. Picture of Agile Manifesto

B. *What is Agile Development?*

Agile approaches are sets of best practices that are appropriate for certain sorts of projects and organizations [8], [9]. It is important to understand that in agile, there are no penalties. There is nothing but feedback. Agile is about improvement, therefore if anything can be better, it must be improved [8]. To be successful, a circle of trust with the product owner and all the stakeholders is needed to be built. They must be convinced that this is the way to work, and they will benefit from this approach. This is achieved by commitment, coordination, and excellent results. Being in time with high quality products is the key for that [8]. Agile development is separated into several types of techniques, each with its own set of best uses. It is crucial to put in mind that not one of these methods can be used in their current form. A mix of a few or all of these may be the optimal development process for the task at hand.

III. AGILE DEVELOPMENT METHODS

Several agile development methods were found in the literature. Comparing with the traditional methodologies, In the last few years, significant inroads have been made into the software industry by agile methodologies [10], [11], [12]. Table I shows a comparison of traditional software development methodologies versus agile software development methodologies.

A. *Scrum*

Scrum is the most known method of agility. Agile is not only scrum, but the agile war has also even been stated to have been won by scrum [8], [13]. In rugby, scrum is an offensive term that refers to a team pushing forward as a single unit in order to score [14]. Scrum is seen as more of a communication methodology than a development one, which depends on the following rules:

- Priorities of the customer: the set of user stories (requirements) they desire. Scrum development is split to 1-4 weak sprints, which are equal to iterations. The Scrum team is a multi-functional group with all of the capabilities required to accomplish the development project.
- A product backlog is the starting point for the Scrum development process (see Fig. 2). The backlog's top items are subsequently pushed to the sprint backlog. During the sprint, the Scrum team is jointly responsible for developing, integrating, testing, and documenting all of the user stories assigned to the sprint.

TABLE 1. COMPARISON OF TRADITIONAL SOFTWARE DEVELOPMENT METHODS AND AGILE SOFTWARE DEVELOPMENT METHODS [12]

Parameter	Traditional Methods	Agile Methods
Adaptability Change	Change Sustainability	Change Adaptability
Development Approach	Predictive	Adaptive
Development Orientation	Process-Oriented	People- Oriented
Project Size	Large	Small/Medium
Planning Scale	Long-term	Short-term
Management Style	Command-and-control	Leadership-and-collaboration
Learning	Continuous Learning while Development	Learning is secondary tool
Documentation	High	Low

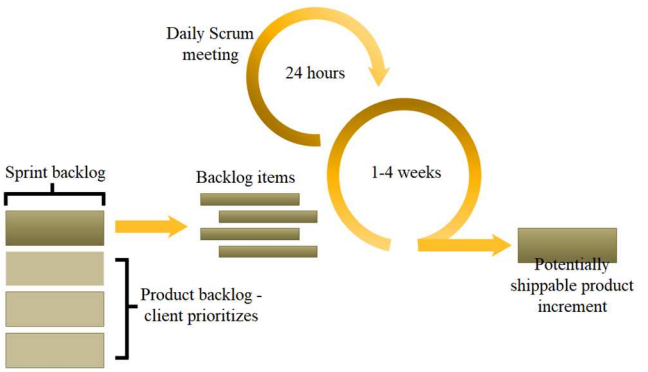

Fig. 2. Development Process of Scrum

B. *Extreme Programming (XP)*

The following is a one-sentence explanation of extreme programming. "Rather than delivering everything you may possibly want at a later date, the extreme programming methodology gives the software you need right now" [15]. XP is a type of iterative development that has proven to be successful in small businesses. Extreme programming prioritizes client satisfaction and constantly strives to deliver software on time (see Fig. 3).

C. *Agile Rational Unified Process Framework (Agile RUP)*

The agile rational unified process (Fig. 4) is built on case-driven development. In other words, it is based on observable user requirements. It employs an iterative method and focuses on the architecture established at an early stage in the process of development. However, the demand of the end user has to be thoroughly integrated in the final documented product is different from other agile processes [16].

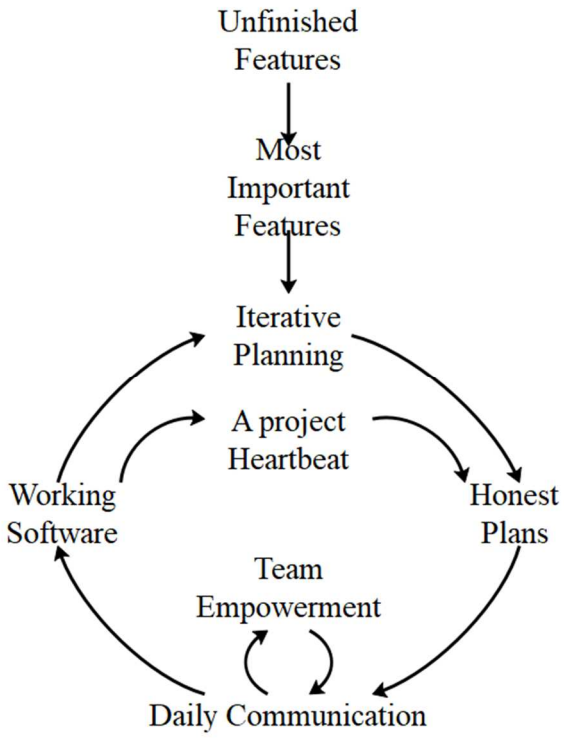

Fig. 3. Extreme Programming [15]

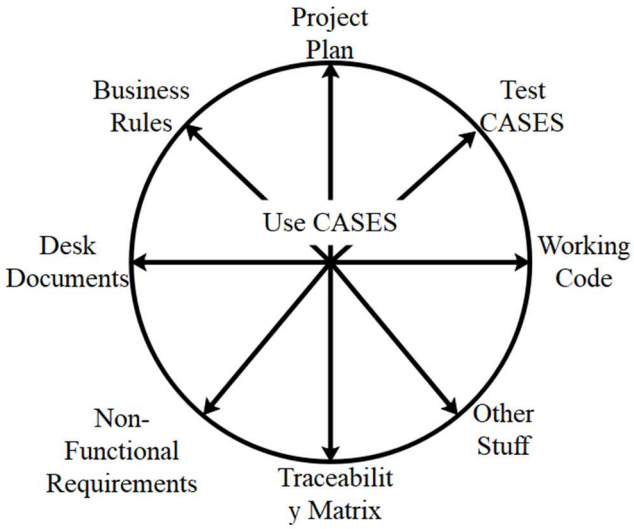

Fig. 4. Agile RUP Picture [8]

D. Feature Driven Development (FDD)

FDD was first introduced in 1997 [17]. More information on FDD was provided in [18], [19]. FDD is a style of agile development which concentrates on two primary stages. These stages are used to establish the features' list and to implement them one by one. The first stage of FDD, which involves identifying the features that will be included in subsequent phases, is critical. The work quality of the first stage determines the precision of project tracking as well as the project code's extensibility and maintainability. Customers must devote 100% of their attention to this stage. The problem domain is represented by UML diagrams, from which the features list is created. Both developers and customers should understand the language used to describe features. The features mentioned in the features list are modest in size, allowing for rapid development of the software application. Work packages are prepared at the start of the implementation process. A work package is a collection of features that are linked together. To complete a work package, one iteration will be necessary. Each iteration usually lasts one to three weeks. Customers are given work packages to test after they are completed [17]. The FDD lifecycle is shown in Fig. 5.

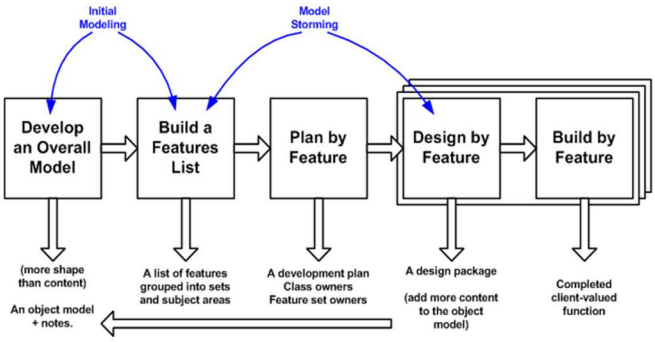

Fig. 5. The FDD Project Lifecycle [19]

E. Feature Driven Development (FDD)

Integrating lean manufacturing ideas into software development gave birth to Lean Software Development (LSD) [20]. LSD is more concerned with principles. LSD is based on the concepts of value stream mapping and attempting to track and eliminate waste. LSD refers to a set of principles derived from Lean manufacturing and implemented to software development. The seven core notions listed in Fig. 6 are the emphasis of these core principles.

Fig. 6. Core Principles of LSD [21]

F. Adaptive Software Development (ASD)

ASD was introduced by James A. Highsmith in 2000 [17]. Instead of the static software development life cycle of plan design build, he proposed a dynamic Speculate collaborate learn software development life cycle [17]. The process begins with the initiation phase of the project, which establishes the development cycle's goals and timelines. In the collaboration phase, several components are being developed at the same time. Components are constantly refined, which is why development cycle planning is an iterative process. It is critical to document the lessons gained about the result's quality from the perspective of the customer, the result's quality from a technical perspective, the procedures of the delivery team and functioning, and the

project's state at the end [22]. ASD project lifecycle is shown in Fig. 7.

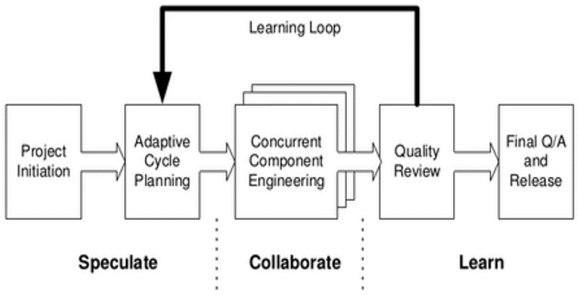

Fig. 7. The ASD Project Lifecycle [22]

G. Kanban System

The Kanban methodology, which belongs to the agile set of techniques, is rapidly gaining favour in the software industry. As pioneered in [23], during the process of software development, Kanban allows you to visualize and limit your work-in-progress. The Kanban technique focuses on work scheduling to aid in the delivery of software products that are just-in-time for implementation. To better simulate business agility, Kanban is being adopted by companies all around the world and is being integrated into their present software development processes. The Kanban technique is distinguished from other Agile-based techniques by a number of characteristics [12]. A Kanban board is a tool for visualizing workflow that allows for job optimization and workflow guidance by categorizing tasks into to-do, in-progress, and completed categories. The Kanban software development method offers workflow efficiency and scheduling, as well as increased team productivity by decreasing idle time. The Kanban technique is directly linked to the continuous delivery of software increments rather than batch releases of functionalities. Customers' dynamic requirements are met by releasing tiny sections of the product in consecutive iterations. Tasks are only performed when they are genuinely necessary under the Kanban methodology. As a result, overproduction is eliminated, as well as wasted work and time. The fundamental goal of Kanban approach is to keep the Work-in-Progress to a minimum in order to optimize the system's workflow in accordance with its capacity. Work-in-progress constraints can be applied to individual workflow steps or the entire process. Kanban system methodology is shown in Fig. 8.

Fig. 8. Kanban System Methodology [24]

H. Dynamic System Development Method (DSDM)

The DSDM provides a framework for developing applications quickly [25]. The feasibility study and the business study are the first two parts of the DSDM. The base requirements are elicited during these two parts. During the development phase, further requirements are elicited. Certain approaches are not required by DSDM. As a result, during the development phase, any RE approach can be employed. Testing is included throughout the lifecycle in DSDM. The DSDM idea is to 'test as you go' [25]. The developer and team members perform all types of testing (technical and functional) in stages. The usage of JAD sessions is specifically emphasized in DSDM, as is prototyping [25], [26].

IV. ADVANTAGES AND DISADVANTAGES OF AGILE METHODOLOGIES

In comparison style, the advantages and disadvantaged of the above-mentioned agile methodologies are presented in Table II.

V. AGILE REQUIREMENT ENGINEERING (AGILE RE)

The field of requirement engineering (RE) arose largely as a result of the rapidly rising size of need specifications, necessitating the development of engineering tools to aid in the discovery of system functionality and restrictions [27]. The developer's management of requirements ends with the delivery of products that meet the criteria of acceptance [28]. Park and Nang confirmed that as the size of a software system grows higher, requirement management becomes more difficult [29]–[31]. Controlling the needs for anticipating and responding to change requests [32]. One of the most important processes in the software delivery and project life-cycle is requirement management [31]. Changes in requirements are welcomed by software development teams that follow the agile software development approach at any point during the product's software development cycle. Agile approaches do not require a lengthy requirements document [26], [31].

RE is formally a part of agile methodologies. On the other hand, the scale of RE is frequently quite small. Documentation, for example, is frequently viewed as a step that slows down the agile process and is thus avoided, according to [26], which makes the tracking of requirements extremely difficult. They also believe that requirement management is not a component of agile development, but that the remaining processes are. Agile methodologies differ when it comes to the implementation of requirement management. Requirements, in scrum, are addressed through user stories. As a result, scrum requirement management is defined as the discussion of user stories that specify genuine requirements. As a result, the product owner takes the lead in software development [33].

User stories and onsite clients are used to address requirements in extreme programming (XP) [34]. User stories with two parts: a written card and conversations following the use of the written card. Written cards are merely 'promises for discussion' [31]. Cards do not have to be finished or even specific. After implementation, Story cards are disposed of [35]. Gather user requirements is

TABLE 2. ADVANTAGES AND DISADVANTAGES OF AGILE METHODOLOGIES

Agile Method	Advantages	Disadvantages
Scrum	- Adaptability and Flexibility - Innovation and Creativity - Time-to-Market - Cost-Cutting - Enhanced Quality - Customer and Employee Satisfaction - Synergy In the Workplace	- Requires Skill and Training - Organizational Transformation - Scalability - Project/Program Management Integration
XP	- Fast - Visible - Reduce costs - Teamwork - Strong relationship with the client	- Code overcomes design - Location - Lack of documentation - Stress
Agile RUP	- It is a complete methodology in itself - Proactive risk resolving - Less integration time - Less development time - Easy online training and tutorial	- It needs expert team members - The process of development is complicated and disorderly - The reuse of components is not possible on cutting edge - Issues with integration in the software development process with large projects and multiple development
FDD	- feature Progress tracking - Multiple teams work - Reduce costs - It has improved process tracking capabilities - It is well-suited to huge groups or projects	- Not ideal for small-size projects - High reliance on one person - Lack of documentation - The approach is designed in such a way that iterations aren't adequately defined by the process
LSD	- The elimination of waste leads to the development process' overall efficiency - Early Delivery - Empowering the development team aids in members' decision-making abilities, resulting in a more motivated team	- The project is highly dependent on cohesiveness of the team and the team members' individual commitments - Needs a highly skilled team - Clients and project sponsors must know the growth of team exactly what they want and make decisions they are prepared to follow through on - In the absence of a competent business analyst, scope creep will inevitably occur - It enables the SRS to progress. However, this creates its own set of issues
ASD	- Make discussion - Get ideas - Make demonstrations - Able to maintain good quality work	- Requires public employees - Time consuming
Kanban	- Easy to learn - Process flexibility - Continuous delivery - Improves the flow of the delivery - Reduces the time it takes for the process to complete	- An obsolete Kanban board that could cause problems during development - Lack of timing - Each phase does not have a timeframe connected with it
DSDM	- A high level of user interaction - Basic functionalities are delivered more frequently and at a faster rate - Projects are completed on schedule and under budget - Provides access by developers to end users	- Not recommended for small businesses or one-time jobs - Because it is a newer model compared to older traditional models such as the waterfall, it is not as well known or understood - When compared to other agile development software methodologies, DSDM might be restricted and difficult to work with due to its strictness and eight principles

depicted in FDD as a UML diagram with a feature list. The feature list is used to keep track of functional requirements and development tasks. The scope of the system and its context are examined at a high level in the solution requirements analysis. For each modelling area, the team evaluates the domain in depth. For each domain, small groups construct a model and present it to their peers for feedback [31].

User requirements are gathered in lean software development by presenting displays to end-users and soliciting feedback. To recognize specific requirements and the environment, the just-in-time production mindset is used. Customer input is initially provided in the form of little cards

or stories. Each card's implementation time is estimated by the developers. Each morning at stand-up meetings, work organization transforms into a self-pulling system. During the speculative phase of Adaptive Software Development (ASD), requirements are gathered. Defining the mission and goals of the project, comprehending constraints, organizing the project, defining and describing requirements, estimating initial scope, and identifying important project risks are the first steps. In a preliminary JAD session, data on project initiation is obtained [36].

User stories in the Kanban system aid in determining what a sprint's actual goals were. One story card is contained in a sprint. A user story is divided into smaller portions by the tasks. A division into server-side and client-side tasks is done to the story. The jobs were broken down into smaller chunks. To keep the project on schedule, developers keep the number of items in a sprint to a minimum [37]. The requirement phase of the agile rational unified process (RUP) entails identifying stakeholders, recognizing the user's issues, developing a basis for estimating, and designing the system's user interface. Activities take place throughout the Inception and Elaboration phases, however, they continue to enhance the design as it progresses throughout the phases. The deliverables are the business use case model. During the construction phase, user stories are implemented and iteratively revised to reflect comprehension of the issue domain as the project progresses.

Finally, there are four requirements management phases in the Dynamic System Development Method (DSDM). Feasibility phase: The requirements for a specific project are collected and evaluated for feasibility and priority. Authors of [38] define five agile Requirements Engineering (RE) models in terms of requirements development based on requirements management methodologies in agile techniques:

1) Input and output RE method: To create an output, combine the inputs.
2) Linear RE method: Similar to the Input and output RE method, however, the 'black box' is now split into various phases.
3) Linear iterative RE method: The iterative technique is the most significant distinction from earlier approaches. This technique works best when requirements must be extremely precise, allowing requirements engineers to repeat the process until all stakeholders are happy.
4) Iterative RE method: This model provides superior version-by-version release support, in comparison to the linear RE method.
5) Spiral model of RE: A complete version of the product is represented by each spiral, and the procedure is carried out in spirals. This model focuses on risk management, something none of the other models do.

Authors of [39] argue that pre-specified requirements are frequently incompatible with agile software development. Thus, there are seven 'best practices' for RE: First, face-to-face communication over written specifications: Instead of documenting requirements first, transfer them directly face-to-face. Advantages: it reduces time wasted and gives the consumer immediate control. Disadvantages: customers must have time available, and when they are acclimated to traditional processes, they tend to distrust agile techniques. Second, iterative RE that during the implementation process, requirements are being developed (which is in line with the agile philosophy). Advantages: because of direct contact with the consumer, good customer relations and requirements are extremely clear. Disadvantages: poor cost/time estimates, a lack of documentation leading to a bad overview, and a lack of attention on non-functional requirements. Third, requirement prioritization goes extreme: Each cycle, prioritize requirements, including additional tasks like bug repairs, with the goal of adding business value rather than expense and risk, as is the case with traditional methodologies. Advantages: each requirement's importance is more evident and does not need to be frozen. Disadvantages: because of continual re-prioritization, the system may become unstable, and non-functional requirements are less significant in the early phases (due to a lack of additional business value), which becomes an issue in the later phases. Fourth, managing requirements change through constant planning: re-adjusting the planning on a regular basis. Advantages: a project that is dynamic in terms of difficulties that develop. Disadvantages: changing the plan isn't always enough; in some cases, the entire process must undergo a redo. Fifth, prototyping that creating prototypes in order to receive rapid feedback from customers. Advantages: feedback from all customers without delay. Disadvantages: scalability and robustness are bad. Sixth, test-driven development that before any features are introduced, tests are run to ensure that they meet highly specific requirements (with a significant level of detail). Advantages: It is possible to combine documentation with existing test code, allowing developers to experiment with new ideas because tests provide rapid feedback. Disadvantages: developers are not used to writing tests first; they need to know a lot of information before they can build good tests. Finally, use acceptance tests and review meetings: Feedback can be created moments after each phase through the use of meetings and tests. Advantages: besides the input, the consumer will gain a thorough understanding of the present development condition, which will build trust. Disadvantages: due to busy schedules, it is often difficult to gather all stakeholders in one place each cycle.

Apart from the fact that adequate requirements are critical for software projects and that Agile projects usually throw some aspects of RE out the window, Authors of [40] offer four points to consider in order to enhance RE's use in Agile projects:

1) Customer interaction: Even within end-user groups, there are frequently divergent opinions that must be identified.
2) Analysis (validation and verification): Validation is common in agile projects, but verification is rare.
3) Non-functional requirements: Because they do not add business value, they are sometimes underrated in projects, yet they are critical to the overall output.
4) Managing change: Requirements Management must be part of agile projects.

Agile requirements engineering approaches can be elicited in a variety of techniques, based on what has been mentioned above. The processes are discussed in the following subsections.

A. Interaction between the Development Team and the Customer

A fundamental feature in all agile methodologies is to have a client accessible or on-site. Authors of [26] emphasizes that the client is involved throughout the development process, but that this does not guarantee that every user or client with the required background is there. It is highly recommended in agile software development to have no communication layers between the development team and the customer. When direct communication between the development team and the customer is made, the chances of miscommunication between the two sides are greatly reduced [41]. Rather than creating comprehensive documentation, the goal of agile requirements engineering is to successfully convey ideas from the customer to the software development team. The time-consuming paperwork and approval processes are no longer required when informal communication between the client and the development team is used. When the requirements change, these things are seen as superfluous. To define client requirements, most firms employ simple methodologies like user stories [39]. When gathering customer requirements, the entire development team should be involved, and the client's common language should be used. Misunderstandings will be less likely as a result of this. Furthermore, if the requirements are too complicated, the customer is urged to break them down into smaller chunks [41].

B. Iterative Requirements Engineering

Functionalities are released in small, regular cycles using agile approaches. This enables the development team to obtain more and more frequent feedback from customers in real time [41]. The development team gains a high-level understanding of the software's important features at the start of the project. High requirements volatility, limited knowledge of the technology utilized, or customers who cannot accurately define the requirements before seeing them are all reasons to not put too much effort into requirements engineering at the start. Agile requirements engineering continues at each development cycle after the initial brief requirements identification. The client and development teams meet at the start of each cycle to discuss the features that must be implemented [39]. As the development progresses, the requirements are incrementally and iteratively detailed [42]. Gradual detailing guarantees that needs are actively worked with at all stages of development, reducing the difficulty of communication gaps both within and between the development and business teams [42]. It is also easier to keep system requirements specifications up to date with an iterative approach to requirements engineering, which leads to a more satisfying customer relationship. When the customer-developer relationship is positive, the consumer will provide feedback to the development team. Iterative requirements engineering allows for frequent client feedback. This aids in the reduction of waste in requirements such as non-essential features. Iterative requirements engineering, according to [39], can be employed in stable contexts when changes in requirements are caused by unforeseen technical challenges.

C. Requirements Prioritization

The highest-priority features are developed first in agile development so that clients can get the most business value [26], [39]. Because the project's understanding grows and new requirements are added during development, the prioritizing should be repeated regularly throughout the process. Authors of [39] and [41] suggested that at the start of each development cycle, the requirements be prioritized. The customer and development team assign priority to each feature to ensure that the most critical requirements are implemented first. They also mentioned that the priority of requirements is based solely on one element, the business value for the customer. They presented a four-step methodology for prioritizing requirements:

The development team calculates how long it will take to implement the feature.

For each functionality, the customer establishes business priorities.

Based on the business priorities, the development team assigns a risk level to each functionality.

The customer and the development team determine which features will be implemented in the iteration.

D. Non-Functional Requirements

Non-functional requirements can be thought of as the limitations that must be met by the entire system. In other words, non-functional requirements describe how the software will perform rather than what it will do [43]. Maintainability, safety, performance, scalability, and portability are examples of non-functional requirements. Non-functional requirements are frequently overlooked in agile methodologies. Customers are frequently more concerned with the core functionality [39]. Customers specify what they want the system to perform, thus they aren't usually concerned with non-functional requirements [26]. However, Customers focusing on some requirements such as interface, simplicity of use and safety is a common exception. Due to the customer's lack of regard for non-functional requirements, the development team should assist the customer in identifying such hidden needs. Authors of [44] also advise that clients and agile team leaders convene meetings to discuss non-functional requirements as early as possible. Authors of [41] also point out that because of the continual connection with the customer, the necessity to express non-functional requirements is less significant in the context of agile software development than in other contexts. After each iteration, the customer can test the program, and if he finds any issues with non-functional needs, the development team can adjust the system to fulfill those requirements in the next iteration. The development team should be aware of the majority of non-functional needs because they can influence the architecture chosen. Regarding non-functional requirements in this manner could hold a significant risk due to a lack of certain techniques to manage them.

E. Documentation in Agile Requirements Engineering

The documentation in agile approaches is low, and the needs are not usually defined [42]. Some agile approaches, on the other hand, advocate for the use of requirements documents; nevertheless, this is mostly dependent on the

development team's decision. When planning the documentation, the size of the team should also be taken into account. When over-documentation is eliminated, the agile team should be cost effective and productive. When the software is updated, the short documentation increases the possibilities of keeping the document up to date. Customers frequently want documentation from the team before the team is resolved, but the scope of this material is relatively limited and focused on the system's basic elements. Despite the fact that modelling is employed as part of agile requirements engineering, the majority of the models will not become part of the system's permanent documentation [26].

F. User Stories

A user story describes the capabilities that a program or system's user or customer finds useful. The user stories are made up of three different elements: a written summary of the plot, Tests that convey and document specifics, as well as conversations about the story. While the story's material may be included in the written description, the detailed information is worked out in dialogues and documented in testing [45]. The written description should be prepared on a piece of paper and should concern a small piece of functionality. After that, the papers are pinned on a board. Because one user story does not contain many specifics, a basic question is where the details are. Cohn responds by saying that many of the requirements can be stated as new stories. It is preferable to have several stories than to have a few major stories [45].

G. Challenges of Minimal Documentation

Agile methodologies, on the whole, tend to produce insufficient documentation. Because documentation is used to transmit information between people in agile teams, a lack of documentation may cause issues. Personnel turnover, quick changes in requirements, a lack of the suitable client representative, or the application's expanding complexity can all cause communication breakdowns. A wide range of issues can occur if communication breaks down. Inability to scale the program, evolve the product over time, or add new people to the development team are examples of issues. A new team member will have a lot of questions about the project, and if he or she has to ask other team members questions all the time, the job will be slowed down [26], [39].

VI. CONCLUSION

To add to the literature on web engineering methodologies in general and agile methodologies in particular, a survey was presented in this paper that give deep insight about the agile methodologies and their advantages and disadvantages. This paper discussed the agile methodologies development methods and summarized the advantages and the disadvantages of these methods. Furthermore, the agile requirements engineering are discussed in detail. This work will be an asset for further work in the field of requirements change management in web engineering projects.

REFERENCES

[1] S. Murugesan and Y. Deshpande, Web engineering: managing diversity and complexity of web application development, ser. Lecture notes in Computer Science 2016. New York: Springer Science and Business Media, 2016.

[2] K. Wakil and D. N. Jawawi, "Comparison between web engineering methods to develop multi web applications," Journal of Software, vol. 12, no. 10, pp. 783–794, 2017.

[3] N. P. de Koch, "Software engineering for adaptive hypermedia systemsreference model, modeling techniques and development process," Thesis, 2001.

[4] S. Aghaei, M. A. Nematbakhsh, and H. K. Farsani, "Evolution of the world wide web: From web 1.0 to web 4.0," International Journal of Web and Semantic Technology, vol. 3, no. 1, pp. 1–10, 2012.

[5] G. Kappel, B. Pr¨oll, S. Reich, and W. Retschitzegger, Web engineering. New York: John Wiley and Sons, 2006.

[6] K. Wakil and D. N. Jawawi, "Model driven web engineering: A systematic mapping study," E-Informatica Software Engineering Journal, vol. 9, no. 1, pp. 107–142, 2015.

[7] D. Batra, "Job-work fit as a determinant of the acceptance of largescale agile methodology," Journal of Systems and Software, vol. 168, p. 110577, 2020.

[8] P. O. Koivisto, "New agile process errors in software development," Thesis, 2010.

[9] K. Schmitz, R. Mahapatra, and S. Nerur, "User engagement in the era of hybrid agile methodology," IEEE software, vol. 36, no. 4, pp. 32–40, 2018.

[10] A. Abdelghany, N. R. Darwish, and H. A. Hefni, "An agile methodology for ontology development," International Journal of Intelligent Engineering and Systems, vol. 12, no. 2, pp. 170–181, 2019.

[11] D. Beerbaum, "Applying agile methodology to regulatory compliance projects in the financial industry: A case study research," Applying Agile Methodology to Regulatory Compliance Projects in the Financial Industry: A Case Study Research (April 26, 2021), vol. JADE, 2021.

[12] G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, "Empirical study of agile software development methodologies: A comparative analysis," ACM SIGSOFT Software Engineering Notes, vol. 40, no. 1, pp. 1–6, 2015.

[13] F. Hayat, A. U. Rehman, K. S. Arif, K. Wahab, and M. Abbas, "The influence of agile methodology (scrum) on software project management," in 2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). IEEE, 2019, pp. 145–149.

[14] The lean approach to agile software development. [Online]. Available: http://www.gatherspace.com/static/agile software development.html

[15] Extreme programming: A gentle introduction. [Online]. Available: http://www.extremeprogramming.org/

[16] Rational unified process (rup) methodology. [Online]. Available: https://techspirited.com/rational-unified-process-rup-methodology

[17] A. F. Chowdhury and M. N. Huda, "Comparison between adaptive software development and feature driven development," in Proceedings of 2011 International Conference on Computer Science and Network Technology, vol. 1. IEEE, 2011, Conference Proceedings, pp. 363–367.

[18] P. Coad, J. d. Luca, and E. Lefebvre, Java modeling color with UML: Enterprise components and process with Cdrom. Upper Saddle River: Prentice Hall PTR, 1999.

[19] S. R. Palmer and M. Felsing, A practical guide to feature-driven development. Upper Saddle River: Pearson Education, 2001.

[20] M. Poppendieck and M. A. Cusumano, "Lean software development: A tutorial," IEEE Software, vol. 29, no. 5, pp. 26–32, 2012.

[21] Lean software development (wave-ii) — 7 lean principles of software development. [Online]. Available: http://www.gatherspace.com/static/agile software development.html

[22] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," VTT Technical Research Centre of Finland Ltd, vol. 478, pp. 1–112, 2017.

[23] D. J. Anderson, Kanban: Successful Evolutionary Change for Your Technology Business. Seattle, USA: Blue Hole Press, 2010.

[24] Lean kanban methodology to application support and maintenance. [Online]. Available: http://www.gatherspace.com/static/agile software development.html

[25] J. Stapleton, DSDM, dynamic systems development method. New York: Addison-Wesley, 1995.

[26] F. Paetsch, A. Eberlein, and F. Maurer, "Requirements engineering and agile software development," in Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for

Collaborative Enterprises, (WET ICE 2003). IEEE, Conference Proceedings, pp. 308–313.

[27] K. Wnuk, Understanding and supporting large-scale requirements management. LU-CS-LIC: 2010-1, Licentiate thesis, 2010.

[28] S. E. M. W. Group, “Requirements management guidebook: Avionics software engineering 4.5.7.” SRM: GDBK: SWELT: 1.0:14AUG98, 1998.

[29] S. Park and J. Nang, “Requirements management in large software system development,” in SMC’98 Conference Proceedings. IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218), vol. 3. IEEE, Conference Proceedings, pp. 2680–2685.

[30] A. Zainol, “Investigation into requirements management practices in the malaysian software industry,” in 2008 International Conference on Computer Science and Software Engineering, vol. 2. IEEE, Conference Proceedings, pp. 292–295.

[31] N. Baruah, “Requirement management in agile software environment,” Procedia Computer Science, vol. 62, pp. 81–83, 2015.

[32] D. J. Reifer, “Requirements management: The search for nirvana,” IEEE Software, no. 3, pp. 45–47, 2000.

[33] M. C. Paulk, “Extreme programming from a cmm perspective,” IEEE Software, vol. 18, no. 6, pp. 19–26, 2001.

[34] K. Beck and M. Fowler, Planning extreme programming. Boston: Addison-Wesley, 2001.

[35] R. Jeffries, A. Anderson, and C. Hendrickson, Extreme programming installed. Boston: Addison-Wesley, 2001.

[36] J. R. Highsmith, Adaptive software development: a collaborative approach to managing complex systems. New York: Addison-Wesley, 2013.

[37] O. Liskin, K. Schneider, F. Fagerholm, and J. M¨unch, “Understanding the role of requirements artifacts in kanban,” in Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering. ACM, Conference Proceedings, pp. 56–63.

[38] Q. K. Shams-Ul-Arif and S. Gahyyur, “Requirements engineering processes, tools/technologies, and methodologies,” International Journal of Reviews in Computing, vol. 2, no. 6, pp. 41–56, 2009.

[39] L. Cao and B. Ramesh, “Agile requirements engineering practices: An empirical study,” IEEE software, vol. 25, no. 1, pp. 60–67, 2008.

[40] A. Eberlein and J. Leite, “Agile requirements definition: A view from requirements engineering,” in Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE’02), Conference Proceedings, pp. 4–8.

[41] A. Sillitti and G. Succi, Requirements Engineering for Agile Methods. Berlin, Heidelberg: Springer, 2005, pp. 309–326.

[42] E. Bjarnason, K. Wnuk, and B. Regnell, “A case study on benefits and side-effects of agile practices in large-scale requirements engineering,” in Proceedings of the 1st Workshop on Agile Requirements Engineering. ACM, Conference Proceedings, pp. 3:1–5.

[43] K. M. Adams, Nonfunctional requirements in systems analysis and design. Cham, Switzerland: Springer, 2015.

[44] A. De Lucia and A. Qusef, “Requirements engineering in agile software development,” Journal of Emerging Technologies in Web Intelligence, vol. 2, no. 3, pp. 212–220, 2010.

[45] M. Cohn, User Stories Applied: For Agile Software Development. Boston: Addison-Wesley Professional, 2004.