

Documentation Practices in Agile Software Development: A Systematic Literature Review

Md Athikul Islam
Department of Computer Science
Boise State University
Boise, ID, USA
mdathikulislam@u.boisestate.edu

Rizbanul Hasan
Department of Computer Science
Boise State University
Boise, ID, USA
rizbanulhasan@u.boisestate.edu

Nasir U. Eisty
Department of Computer Science
Boise State University
Boise, ID, USA
nasireisty@boisestate.edu

Abstract—Context: Agile development methodologies in the software industry have increased significantly over the past decade. Although one of the main aspects of agile software development (ASD) is less documentation, there have always been conflicting opinions about what to document in ASD. **Objective:** This study aims to systematically identify what to document in ASD, which documentation tools and methods are in use, and how those tools can overcome documentation challenges. **Method:** We performed a systematic literature review of the studies published between 2010 and June 2021 that discusses agile documentation. Then, we systematically selected a pool of 74 studies using particular inclusion and exclusion criteria. After that, we conducted a quantitative and qualitative analysis using the data extracted from these studies. **Results:** We found nine primary vital factors to add to agile documentation from our pool of studies. Our analysis shows that agile practitioners have primarily developed their documentation tools and methods focusing on these factors. The results suggest that the tools and techniques in agile documentation are not in sync, and they separately solve different challenges. **Conclusions:** Based on our results and discussion, researchers and practitioners will better understand how current agile documentation tools and practices perform. In addition, investigation of the synchronization of these tools will be helpful in future research and development.

Index Terms—Software Engineering; Agile Software Development; Documentation; Systematic Literature Review

I. INTRODUCTION

Software documentation is an integral part of software development. It works as a communication medium between developers of a team and is utilized as an information repository by maintenance engineers [76]. Documentation elucidates how the system is structured, its functionalities, and the design rationale [70]. Software documentation should be included as part of software development and is sometimes called “common sense” [26]. Even outdated document serves a purpose and may be helpful [26]. However, the incomplete, wrong, clumsy, abstruse, outdated, and inadequate document often leads to the unpopularity of software documentation among the developers [4]. Regardless of the application type, almost all medium to large software projects produces a certain amount of documentation [76].

ASD is an iterative approach that helps teams deliver value to their customers faster and more efficiently. It is immensely applied in both industry and academia [54]. This incremental approach maintains a strong focus on project goals

and customer involvement. Over the past decade, numerous software developers have adopted the ASD model [33]. This concept of agile is gained by different agile methodologies like Scrum, eXtreme Programming (XP), Crystal, Lean, Dynamic Systems Development Method (DSDM), and Feature-Driven Development (FDD). Documentation has a lower priority than working software in Agile practices [27]. Some agile practitioners consider code as its documentation. As a result, the information that is recorded in documentation (documented information, henceforth) may not be well maintained, resulting in inadequate information for the team to understand development tasks [40]. Another reason for less focus on documentation is that it consumes time that could have been allocated in development [70].

As modern software systems are complicated, developers revisit the system more often. Trivial maintenance work is assigned to junior developers who have little experience with the code. As a result, lack of documentation hinders inter-team building and knowledge loss [78]. In addition, users of a software product expect quality results [2]. Considering these scenarios, agile documentation plays a significant role in ASD [70], [71]. Moreover, offshore agile development is widespread nowadays, and documentation is one of the key factors to make offshore agile development successful [58]. Therefore, developers should document the features of each iteration to help future developers refactor them into smaller tasks.

The practitioners working on agile documentation have implemented different documentation strategies to overcome the issues raised by the lack of documentation. They have identified different key elements that developers should document such as user stories, functional requirements, source code, etc [10], [13], [43]. They have also developed various tools and models such as wikis, simul loco, doxygen, etc to document these elements effectively [9], [55], [66]. In this paper, we focus on understanding the pivotal information to document in agile and the existing techniques and tools that result in document optimization. We conducted a systematic literature review from existing research and analyzed them to answer our research questions mentioned in section II. Our findings will benefit any agile practitioners and developers to optimize their documentation effort and help researchers in further study.

II. RESEARCH METHODOLOGY

In our research methodology, we followed the directions proposed by Kitchenham and Charters [45]. We have divided our research review into four steps, namely planning, conducting, and reporting the review results.

A. Planning

We have planned this review by confirming the need for such a review and have proposed our research questions accordingly. Our planning phase includes search strategy, search string, and inclusion/exclusion criteria.

1) *Research questions*: We pose the following research questions to drive this study.

- **RQ1: Which information do agile practitioners document?**
- **RQ2: Which documentation generation tools and methods do agile practitioners use?**
- **RQ3: How can the tools and methods overcome the documentation challenges in agile software development?**

2) *Search strategy*: After defining the need for this systematic review and research questions, we started to carry out the formulation of a search strategy based on the guidelines provided by Kitchenham and Charters [45]. In Table I we broke down the question into individual facets i.e. population, intervention, and constructed search string using Boolean ANDs and ORs. We initially fetched studies from the electronic databases and then explored them through reference searches (snowballing) to seek other meaningful studies. After that, we applied our inclusion and exclusion criteria to the fetched studies involving a different number of researchers, as explained in Section II-A4.

3) *Search criteria*: The search criteria used for this review consist of three parts - C1, C2, and C3, defined as follows:

- C1: We constructed the C1 string, which enables the keyword agile either in the title or abstract.
- C2: The C2 string is made up of keywords such as document or document* either in title or abstract.
- C3: We constructed the C3 part, which enables keywords such as tools or document* tools either in the title or abstract.

The boolean expression: C1 AND C2 OR C3

We provided our search string in Table I.

Another key thing to note is that we have filtered out the result fetched from the search query by applying the checkbox feature of the IEEE Xplore database. In this case, we filtered out publication topics such as internet, organizational aspects, computer-aided instruction, DP industry, computer science education, educational courses, knowledge management, mobile computing, security of data, business data processing, and teaching not relevant to our research.

4) *Inclusion and exclusion criteria*: As per the guidelines of Kitchenham and Charters [45] we have set inclusion and exclusion criteria based on our research questions. Here, we only considered papers that are in English, published in

TABLE I
SEARCH STRING

Search string
((("Document Title": "agile" OR "Abstract": "agile") AND ("Document Title": "document*" OR "Abstract": "document*")) OR ("Document Title": "document* tools" OR "Abstract": "document* tools"))

conferences and journals, and published within the time frame 2010 - 2021. The published papers should describe the agile documentation approach, tools, or knowledge relevant to our RQs. Therefore, we did not include any opinion, viewpoint, keynote, discussions, editorials, comments, tutorials, prefaces, anecdote papers, and presentations. In addition, we excluded the papers that did not discuss agile documentation or agile documentation tools but may discuss agile software development methods as a side topic.

B. Conducting the review

Once we agreed on the protocol, we started our review properly. This section discusses the findings of our search and extracted data from relevant databases and sources.

1) *Study search and selection*: We searched the IEEE Xplore database against our search query and search criteria and fetched 206 studies. In round 1, the first author immensely analyzed the titles and abstracts of the fetched studies based on the inclusion criteria. After the first round, we came out with 77 papers, and most of these studies covered all the inclusion criteria. A critical part was to ensure the papers did not come from opinions, discussions, editorials, comments, tutorials, prefaces, and presentations as per the exclusion criteria. In round 2, we inspected the full-text review of the papers based on all inclusion and exclusion criteria. We read the papers fully a few times where there were disagreements and required consensus. We excluded 23 papers based on the exclusion criteria. Finally, to satisfy the inclusion of relevant primary studies as much as possible, we performed backward and forward snowballing following the guideline provided by Wohlin [84] and included 20 more papers in our list of primary studies. The final pool of selected papers was 77.

2) *Data extraction*: We followed the data extraction strategy of Kitchenham and Charters [45] and came up with a data extraction form that we designed to collect all the information needed to address the review questions. We set a few quality evaluation criteria, such as

- How well was data collection carried out?
- Is the research design defensible?
- How much are the findings credible?
- Is the research scope well addressed?

In addition to the RQs and quality evaluation criteria, the form included the data as (i) name of the reviewer, (ii) date of data extraction, (iii) title, (iv) authors, (v) journal, (vi) publication details, (vii) future work, (viii) limitations, (ix)

year of publication, (x) methodology, (xi) data analysis, (xii) validation technique, (xiii) relevancy and xiv) space for additional notes. This data extraction was performed independently by the first and second authors.

3) *Data synthesis*: After the data extraction, we combined and summarized the results of the included primary studies according to the guidelines of Kitchenham and Charters [45]. Our data synthesis includes both quantitative and descriptive. For the descriptive synthesis, we tabulated the data based on research questions. In this case, we synthesized what type of information and tools are used in agile documentation. On the other hand for quantitative synthesis, we again developed a tabular form for research questions. These tables were structured to highlight similarities and differences between study outcomes. Later the data of these tables were represented by bar charts and pie charts.

III. RESULTS

In this section, we present our findings. We address each research question from RQ1 to RQ3.

A. (RQ1) Which information do agile practitioners document?

Table II summarizes our findings of documented information. Our primary pool of studies consisted of interviews, surveys, case studies, experiments, and statistical analysis. Many agile practitioners, graduate students, and software engineers directly participated in these studies. We collected the findings from these studies and grouped them. The following sections briefly describe each element of Table II.

1) *User stories*: User stories function as the shortcut for more formal documentation and require more details [29]. On the other hand, they represent the small, concise user-driven features and hence need to be documented [56].

2) *Functional requirements*: End-users define these requirements and expect them as facilities when they interact with the system. Therefore, these functionalities focus more on the technical aspects that need to be implemented [57].

3) *Non-functional requirements*: The success of an agile project depends on the non-functional requirements, which are also referred to as quality requirements [12]. These requirements are quality constraints that must be satisfied by the system. So, failure to meet these requirements compromises the entire system and makes it useless [23].

4) *Source code*: The source code should be documented for traceability and deriving the code that does not perform [10].

5) *UI structure*: These are wireframes and sometimes are documented externally. However, they can be the basic layouts of application screens and the product owner usually provides these wireframes [13].

6) *Technical debt*: If a system needs fixes or updates, it is best to document them currently. Similarly, it is best to document them in the case of technical debt. As a result, future developers will be aware of that technical debt. Therefore, all instances of technical debt should be documented [36].

7) *System architecture*: Some documentation tools have evolved to document architecture [10], [31]. Architecture works as the backbone of the entire system [10]. Documenting architecture is one of the most complicated and challenging parts of software development [31].

8) *API reference guides*: To understand the usage and integration of API, APIs should have comprehensive documentation. Good API reference guides make the APIs easier to maintain and helps onboard new developers to the team [75].

9) *Test specification*: It is tough to demonstrate large-scale systems solely using test cases in the industry. Testing needs more mature documentation to keep track of the test cases, user scenarios, and bugs [10].

B. (RQ2) Which documentation generation tools and methods do agile practitioners use?

In order to answer this research question, we first explored the current tools and methods used in ASD from our primary pool of studies and found a total of 23 tools and methods. Next, we categorized them based on their document type and found ten categories. Table III lists categories alongside their tools and Figure 1 shows the percentage of tools under each category. We also listed the studies that reported the tools and methods. Moreover, we mentioned the role of each tool or method in the right-most column of the table.

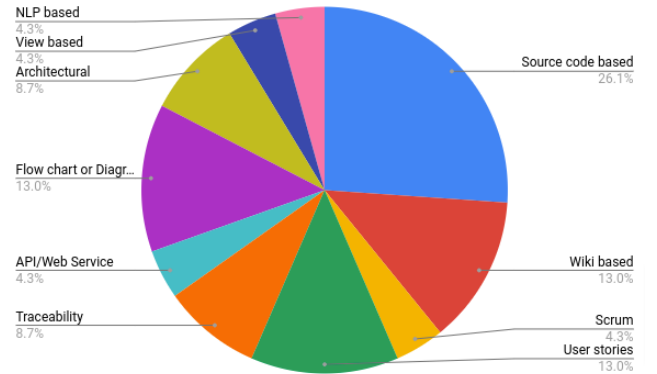


Fig. 1. Categorisation of documentation tools/methods

1) *Source code based documentation*: Source code-based documentation can mitigate certain risks [32]. This category consists of 6 tools, and these tools mainly focus on how software practitioners can generate documentation from source code comments. These tools cover the functionalities of some of the popular documentation generation tools like JavaDoc or Docstring and offer some additional features [44].

2) *Wiki based documentation*: When developers consider straightforward and flexible documentation options in agile, they consider wiki-based documentation in the first place because the primary goal of the wiki is to minimize the development–documentation gap by making documentation more convenient and attractive to developers [80]. For example, sprintDoc and XSDoc are tools based on wikis and can be integrated with other tools such as the VCS IDE.

TABLE II
KEY ELEMENTS TO ADD IN AGILE SOFTWARE DOCUMENTATION

No.	Information to document	Studies that reported the information
1	User stories	[10], [13], [20], [28], [30], [34], [37], [56], [57], [65], [78], [82]
2	Functional requirements	[10], [31], [57], [65], [71]
3	Non-functional requirements	[10], [12], [13], [23], [37], [47], [81]
4	Source code	[10], [43], [44], [55], [66], [83]
5	UI structure	[13], [34], [37], [57], [78]
6	Technical debt	[12]
7	System architecture	[10], [28], [31], [78]
8	API reference guides	[38], [57], [75]
9	Test specification	[10], [1], [71]

3) *Scrum*: Scrum is a lightweight framework for agile development and is very popular. Scrumconix supporting scrum proved to be a valuable and lightweight tool to document and understand a software project [59].

4) *User story*: The user stories explain how the software will work for the users and provide an essential source for the design of the software according to user needs [1]. Methods such as the COSMIC method [20], [65] can measure the quality of user stories to generate high-quality documentation.

5) *Traceability*: Traceability tools like Trace++ support the transition from traditional to agile methodologies. They offer traceability between documents generated during conventional software development and agile methods [28]. In addition, TraceMan provides traces to critical agile artifacts [10].

6) *API/Web service*: In this category, we found a tool called Docio which can generate API documents with I/O examples [38]. This tool is more like the popular REST API documentation generation tool Swagger but only supports the C programming language [79].

7) *Flow chart*: A few tools emphasize providing meaningful graphical diagrams either based on source code or requirements [46], [69]. Flowgen, CLARET, and TCC are some of the tools in this category.

8) *Architectural*: Experts ascertained the lack of documentation and architectural design in agile projects [60]. Abstract specification tool in this category assists the architects in organizing relevant information regarding the architecture while creating design and architecture blueprints, thus reducing the effort of documentation [31]. Also, Active Documentation Software Design (ADSD) is an approach that enables incorporating domain documentation to agile development, while having the processes adaptive [67].

9) *View based*: View-based software documentation enables different perspectives on the software system and enables the explicit and simultaneous modeling of all of those viewpoints as views in the documentation [11].

10) *NLP-based*: Researchers are aware of integrating modern NLP-based techniques and tools into the source code comments where documentation is only available in the form of source code comments. As a result, these tools directly contribute to determining the quality of documentation.

JavadocMiner is one of such NLP-based tools that developers can easily embed with Eclipse IDE [83].

C. (RQ3) How can the tools and methods overcome the documentation challenges in agile software development?

Agile methods or tools that have tried to address the challenges in dynamic contexts have gained much interest among practitioners and researchers [11]. Keeping that in mind, different researchers attempted to identify those challenges and built tools that provide on-demand solutions [25]. We listed all agile challenges in table IV that our previously mentioned tools and methods attempted to resolve. Figure 2 represents a number of tools/methods that resolved a particular challenge.

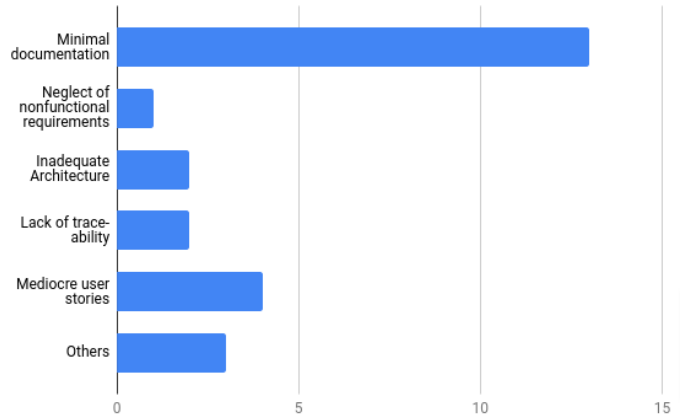


Fig. 2. Number of tools/methods resolving a particular challenge

1) *Minimal documentation*: One of the primary challenges in agile documentation is to keep the documentation minimal [16], [21]. Many documentation generation tools that generate documentation from source code and chart, diagram, and flowchart-based documentation evolved to keep documentation minimum and simple. Practitioners must keep minimal documentation to enhance agile software products, and the tool simul loco comments may answer this problem. Simul loco documentation is extremely useful [66]. Moreover, GitHub plus Markdown support options so that reviewers can give a quick review, and it only takes a few minutes for a minor

TABLE III
TOOLS AND METHODS USED IN AGILE SOFTWARE DOCUMENTATION

No.	Documentation type	Tools/Methods	Studies that reported the practice	Role
1	Source code based documentation	simul loco	[66]	Creates a graphical comment layer over source code that can contain any resource for documentation [66].
		doxygen	[55], [44]	JavaDoc or docstring like documentation tool for Java, C++, Python, and other languages [55].
		JavadocMiner	[44], [83], [43]	Wrapper of JavaDoc and provides quality assessments and recommendations on how Javadoc comments can be improved [44].
		GitHub plus mark-down	[48]	Agile 'Docs Like Code' solution that puts documentation close to the code while software development tools and techniques are applied [48].
		Graphical UML class models	[14], [77], [72], [23], [15], [53], [51], [64]	Graphical UML class models for source code in continuous agile development [14].
		XML_DocTracker	[8]	Produces the software requirements specification (SRS) from the source codes if the SRS did not exist for that particular software [8].
2	Wiki based	Wikis	[9], [73], [41]	Provides a collaborative environment that people use to co-author HTML-based information [9].
		sprintDoc	[80]	Works with issue tracker, wikis, VCS, IDE [80].
		XSDoc	[80], [5], [6]	Minimizes the development–documentation gap by making documentation more user-friendly and attractive to developers [5].
3	Scrum	Scrumconix	[59]	Composition of scrum and ICONIX methods that uses a lightweight approach to document in AGSD environments [59].
4	User stories	COSMIC method	[20], [24]	Assesses the quality of the documentation by analyzing how functional processes are documented in the requirements, such as in the user stories [20].
		CMMI	[3], [52]	Divides the goals into two parts and emphasizes that the product must be delivered on time and meet all of the specified standards [3].
		LAQF	[42]	Lightweight document oriented reusable agile quality framework [42].
5	Traceability	TraceMan	[10]	Traces among user stories, traditional requirements documents, test specifications, architecture design, and source code [10].
		Trace++	[28]	Inherits traditional traceability relationships in order to support the transition from traditional to ASD [28].
6	API/Web Service	Docio	[38]	Documents API Input/Output with parameter and response type [38].
7	Flow chart or Diagrams	Flowgen	[46]	Generates flowcharts from annotated C++ source code and high-level UML activity diagrams, one for each function or method in the C++ sources [46].
		CLARET	[39]	Facilitates functionality to create the use case specifications using natural language [39].
		Ticket-commit network chart (TCC)	[68], [69]	Visually represents time-series commit activities with issued tickets [68].
8	Architectural	Abstract specification tool	[31]	Contains the most relevant and essential information on the architecture solution [31].
		Active Documentation Software Design (ADSD)	[67]	Provides an architectural design in which domain knowledge is represented explicitly and is isolated from other segments of code [67].
9	View based	View-based software documentation	[11]	Utilizes existing software modeling techniques and improves the current methods of software documentation [11].
10	NLP based	JavadocMiner	[83]	Provides a complete environment for embedding NLP into software development [83].

TABLE IV
THE SOLUTION PROVIDED BY TOOLS AND METHODS TO CERTAIN CHALLENGES

No.	Challenge	Description	Tools/Methods	Solutions
1	Minimal documentation [16], [21], [47]	The lack of documentation imposes a variety of problems including the inability to scale the software and add new members. [16]	Source code based documentation (simul loco [66], doxygen [44], [55], JavadocMiner [43], [44], [83], GitHub plus markdown [48], Graphical UML class models [14], [15], [23], [51], [53], [64], [72], [77], XML_DocTracker [8]), Wiki based (Wikis [9], [41], [73], sprintDoc [80], XSDoc [5], [6], [80]), Flow chart or Diagrams (Flowgen [46], CLARET [39], Ticket-commit network chart (TCC) [68], [69]), Abstract specification tool [31]	Generates minimal documentation with a very little effort. The documentation can be either generated from the source code comments or XML or YAML or JSON file.
2	Neglect of non-functional requirements [16], [22], [47], [49], [74]	Customers often prioritize core functionality over non-functional requirements (NFRs) such as scalability, maintainability, portability, safety, or performance. [16], [50]	TraceMan [10]	Enables traceability among non-functional requirements.
3	Inadequate Architecture [7], [63]	As further requirements are known, the architecture designed by the development team during the initial phases may become outdated or inadequate. [63]	Architectural (Abstract specification tool [31], Active Documentation Software Design (ADSD) [67])	Generate architecture design with relevant and updated information.
4	Lack of traceability [18], [35], [61]	Tracing becomes challenging when dealing with large scale or distributed software development efforts. [18]	Traceability (TraceMan [10], Trace++ [28])	Trace appropriate user stories or different artifacts even if the product scales in the future.
5	Mediocre user stories [17], [19], [74]	Lack of proper user stories leads to failure to achieve an overview of the product for team members. [19], [62]	User stories based (COSMIC method [20], [24], CMMI [3], [52], LAQF [42]), TraceMan [10]	Measure the quality of user stories and help to write high-quality user stories.
6	Others [18]		API/Web Service (Docio [38]), View based (View-based software documentation [11]), Scrum (Scrumconix [59])	Generate API documentation from JSON file or supports view based software documentation.

update. A document can be improved easily and continuously [48]. Abstract specification tool proposes a considerably shorter abstract specification document, requiring minimal documentation efforts and resulting in shorter documentation that is easier to review, update, and communicate [31].

2) *Neglect of non-functional requirements*: The effect of requirements changes on the architecture is crucial. It was difficult to trace precisely which architectural decisions had to be reconsidered because of the lack of traceability between the textual requirements specification documents and the architectural models. TraceMan fixes this since the trace links are created during the artifacts' creation. As a result, we can better understand the functional and non-functional requirements using TraceMan more accurately and consequently [10].

3) *Inadequate Architecture*: The primary goals of agile development are flexibility, minimalism, and collaboration. Abstract specification tool achieves these by creating a short and focused architecture document [31].

4) *Lack of traceability*: Conventional agile projects entail intensive labor work to generate and maintain traceable links. Consequently, lack of traceability provides a weak layer over the software system no matter how much flexible the system is [18]. On the other hand, trace++ generates a large number

of traceability relations combining the various artifacts [28].

5) *Mediocre user stories*: Although user stories are essential for ASD, people struggle to document high-quality user stories. Even user stories mentioned in the current dataset are of poor quality [19]. TraceMan produces high-quality user stories by having detailed traces of user store information [10].

6) *Others*: Some tools resolve the complex API documentation challenges by creating API documentation with ease [38]. In addition, there are some tools such as Scrumconix [59], and view-based software documentation that cover challenges posed in the area of scrum and view-based [11].

IV. THREATS TO VALIDITY

The threats to our systematic literature review are the specification of the candidate pool of papers and primary study selection bias. We selected our primary pools of studies through database searches and used keywords. Our keywords were very precise, and we obtained a good number of papers. However, we may missed some papers due to our specific search string.

We also used a specific period to select our studies, which might have discarded relevant papers. On the other hand, we relied on IEEE Xplore for a primary pool of studies,

which threatens to have a complete set of primary studies. To mitigate this risk, we performed both backward and forward snowballing, which eventually resulted in a collection of papers from other databases like ACM, Springer, etc. We followed the standard inclusion and exclusion criteria, which might still introduce some personal bias.

V. CONCLUSION

Working software gets priority over detailed documentation in agile software development. Even though documentation is less of a priority in ASD, studies have shown that a minimal level of documentation is essential. This research aimed to identify key elements to record in ASD and locate appropriate tools to aid in documentation. We conducted a systematic literature review to identify essential information in agile documentation and the effectiveness of current methodologies and tools in agile software development. As a result, we have compiled a list of essential elements in agile documentation and tools and approaches that can help alleviate the documentation challenges.

Our findings will aid in understanding key aspects of agile documentation and how agile documentation tools and approaches function. In the future, we want to map the relationships between these technologies and develop a method that can be used as a one-stop solution for agile documentation. We also intend to conduct a multi-vocal literature review to find more industry concerns and solutions. Finally, our future plan also involves a survey of agile practitioners to see the usefulness of this article.

REFERENCES

- [1] . Iso/iec/ieee intl. standard - systems and software engineering – developing user documentation in an agile environment. *ISO/IEC/IEEE 26515 First edition 2011-12-01; Corrected version 2012-03-15*, pages 1–36, 2012.
- [2] . Iso/iec/ieee intl. standard - systems and software engineering — developing information for users in an agile environment. *ISO/IEC/IEEE 26515:2018(E)*, pages 1–32, 2018.
- [3] S. K. Aggarwal, V. Deep, and R. Singh. Speculation of cmmi in agile methodology. In *2014 Intl. Conf. on Advances in Computing, Communications and Informatics (ICACCI)*, pages 226–230, 2014.
- [4] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza. Software documentation issues unveiled. In *Intl. Conf. on Software Engineering (ICSE)*, 2019.
- [5] A. Aguiar and G. David. Wikikiwiki weaving heterogeneous software artifacts. In *Intl. Symposium on Wikis, WikiSym '05*, page 67–74, 2005.
- [6] A. Aguiar, G. David, and M. Padilha. Xsdoc: an extensible wiki-based infrastructure for framework documentation. In *JISBD*, 2003.
- [7] F. Akbari and S. M. Sharafi. A review to the usage of concepts of software architecture in agile methods. In *Intl. Symposium on Instrumentation Measurement, Sensor Network and Automation (IMSNA)*, 2012.
- [8] H. Aman and R. Ibrahim. Xml_doctracker: Generating software requirements specification (srs) from xml schema. In *2016 Intl. Conf. on Information Science and Security (ICISS)*, pages 1–5, 2016.
- [9] S. Ambler. *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.
- [10] P. O. Antonino, T. Keuler, N. Germann, and B. Cronauer. A non-invasive approach to trace architecture design, requirements specification and agile artifacts. In *23rd Australian Software Engineering Conf.*, 2014.
- [11] J. Bayer and D. Muthig. A view-based approach for improving software documentation practices. In *Intl. Symposium and Workshop on Engineering of Computer-Based Systems (ECBS'06)*, 2006.
- [12] W. Behutiye, P. Rodríguez, M. Oivo, S. Aaramaa, J. Partanen, and A. Abhervé. How agile software development practitioners perceive the need for documenting quality requirements: a multiple case study. In *2020 46th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA)*, pages 93–100, 2020.
- [13] W. Behutiye, P. Seppänen, P. Rodríguez, and M. Oivo. Documentation of quality requirements in agile software development. In *Evaluation and Assessment in Software Engineering, EASE '20*, page 250–259, 2020.
- [14] E. Braude. Incremental uml for agile development: Embedding uml class models in source code. In *2017 IEEE/ACM 3rd Intl. Workshop on Rapid Continuous Software Engineering (RCoSE)*, 2017.
- [15] E. Braude and J. Van Schooneveld. Poster: Incremental uml for agile development with prexel. In *2018 IEEE/ACM 40th Intl. Conf. on Software Engineering: Companion (ICSE-Companion)*, 2018.
- [16] L. Cao and B. Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(1):60–67, 2008.
- [17] F. E. Castillo-Barrera, M. Amador-García, H. Pérez-González, and F. E. Martínez-Pérez. Agile evaluation of the complexity of user stories using the bloom's taxonomy. In *2017 Intl. Conf. on Computational Science and Computational Intelligence (CSCI)*, pages 1047–1050, 2017.
- [18] J. Cleland-Huang. *Traceability in Agile Projects*, pages 265–275. Springer London, London, 2012.
- [19] F. Dalpiaz and S. Brinkkemper. Agile requirements engineering with user stories. In *2018 IEEE 26th Intl. Requirements Engineering Conf. (RE)*, pages 506–507, 2018.
- [20] J.-M. Desharnais, B. Kocaturk, and A. Abran. Using the cosmic method to evaluate the quality of the documentation of agile user stories. In *Intl. Workshop on Software Measurement and the 6th Intl. Conf. on Software Process and Product Measurement*, 2011.
- [21] A. Deuter. Slicing the v-model – reduced effort, higher flexibility. In *2013 IEEE 8th Intl. Conf. on Global Software Engineering*, 2013.
- [22] D. Domah and F. J. Mitropoulos. The nerv methodology: A lightweight process for addressing non-functional requirements in agile software development. In *SoutheastCon 2015*, pages 1–7, 2015.
- [23] S. Dragicevic, S. Celar, and L. Novak. Use of method for elicitation, documentation, and validation of software user requirements (medov) in agile software development projects. In *Intl. Conf. on Computational Intelligence, Communication Systems and Networks*, pages 65–70, 2014.
- [24] J.-F. Dumas-Monette and S. Trudel. Requirements engineering quality revealed through functional size measurement: An empirical study in an agile context. In *Intl. Workshop on Software Measurement and the Intl. Conf. on Software Process and Product Measurement*, 2014.
- [25] W. R. Fitriani, P. Rahayu, and D. I. Sensuse. Challenges in agile software development: A systematic literature review. In *2016 Intl. Conf. on Advanced Computer Science and Information Systems (ICACIS)*, 2016.
- [26] A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering, DocEng '02*, 2002.
- [27] M. Fowler, J. Highsmith, et al. The agile manifesto. *Software development*, 9(8):28–35, 2001.
- [28] F. Furtado and A. Zisman. Trace++: A traceability approach to support transitioning to agile software engineering. In *2016 IEEE 24th Intl. Requirements Engineering Conf. (RE)*, pages 66–75, 2016.
- [29] W. Gerard, S. Overbeek, and S. Brinkkemper. Fuzzy artefacts: Formality of communication in agile teams. In *2018 11th Intl. Conf. on the Quality of Information and Communications Technology (QUATIC)*, 2018.
- [30] D. D. Gregorio. How the business analyst supports and encourages collaboration on agile projects. In *Intl. Systems Conf. SysCon 2012*, 2012.
- [31] I. Hadar, S. Sherman, E. Hadar, and J. J. Harrison. Less is more: Architecture documentation for agile development. In *2013 6th Intl. Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2013.
- [32] M. Hammad, I. Inayat, and M. Zahid. Risk management in agile software development: A survey. In *2019 Intl. Conf. on Frontiers of Information Technology (FIT)*, pages 162–1624, 2019.
- [33] P. Heck and A. Zaidman. A framework for quality assessment of just-in-time requirements: the case of open source feature requests. *Requirements Engineering*, 22(4):453–473, 2017.
- [34] A. Hess, P. Diebold, and N. Seyff. Towards requirements communication and documentation guidelines for agile teams. In *2017 IEEE 25th Intl. Requirements Engineering Conf. Workshops (REW)*, 2017.
- [35] N. N. Hidayati and S. Rochimah. Requirements traceability for detecting defects in agile software development. In *2020 10th Electrical*

- [36] J. Holvitie, S. A. Licorish, R. O. Spínola, S. Hyrynsalmi, S. G. MacDonell, T. S. Mendes, J. Buchan, and V. Leppänen. Technical debt and agile software development practices and processes: An industry practitioner survey. *Information and Software Technology*, 2018.
- [37] A. Jarzębowicz and K. Polocka. Selecting requirements documentation techniques for software projects: A survey study. In *2017 Federated Conf. on Computer Science and Information Systems (FedCSIS)*, 2017.
- [38] S. Jiang, A. Armaly, C. McMillan, Q. Zhi, and R. Metoyer. Docio: Documenting api input/output examples. In *2017 IEEE/ACM 25th Intl. Conf. on Program Comprehension (ICPC)*, pages 364–367, 2017.
- [39] D. N. Jorge, P. D. L. Machado, E. L. G. Alves, and W. L. Andrade. Integrating requirements specification and model-based testing in agile development. In *2018 IEEE 26th Intl. Requirements Engineering Conf. (RE)*, pages 336–346, 2018.
- [40] R. Kasauli, G. Liebel, E. Knauss, S. Gopakumar, and B. Kanagwa. Requirements engineering challenges in large-scale agile system development. In *Intl. Requirements Engineering Conf. (RE)*, 2017.
- [41] R. Kavitha and M. Irfan Ahmed. A knowledge management framework for agile software development teams. In *2011 Intl. Conf. on Process Automation, Control and Computing*, pages 1–5, 2011.
- [42] M. A. Khalid, T. Anees, and A. Moeed. Laqf: Lightweight document oriented, reusable agile quality framework. In *2019 Intl. Conf. on Innovative Computing (ICIC)*, pages 1–8, 2019.
- [43] N. Khamis, J. Rilling, and R. Witte. Assessing the quality factors found in in-line documentation written in natural language: The javadocminer. *Data & Knowledge Engineering*, 87:19–40, 2013.
- [44] N. Khamis, R. Witte, and J. Rilling. Automatic quality assessment of source code comments: The javadocminer. In C. J. Hopfe, Y. Rezugui, E. Métais, A. Preece, and H. Li, editors, *Natural Language Processing and Information Systems*, pages 68–79, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [45] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. 2007.
- [46] D. A. Kosower, J. J. Lopez-Villarejo, and S. Roubtsov. Flowgen: Flowchart-based documentation framework for c++. In *2014 IEEE 14th Intl. Working Conf. on Source Code Analysis and Manipulation*, 2014.
- [47] M. Käpyaho and M. Kauppinen. Agile requirements engineering with prototyping: A case study. In *2015 IEEE 23rd Intl. Requirements Engineering Conf. (RE)*, pages 334–343, 2015.
- [48] L. Lee. Extended abstract: Documentation development practice in open source startups - take pingcap as an example. In *2019 IEEE Intl. Professional Communication Conf. (ProComm)*, pages 255–256, 2019.
- [49] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, predicting and prioritizing (cepp) non-functional requirements metadata during the early stages of agile software development. In *SoutheastCon 2015*, 2015.
- [50] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, and prioritizing (cepp) nfrs in agile software engineering. In *SoutheastCon 2017*, 2017.
- [51] D. Matheson. Modeling requirements: The customer communication. In *2014 IEEE 5th Intl. Workshop on Requirements Prioritization and Communication (RePriCo)*, pages 15–24, 2014.
- [52] J. R. Miller and H. M. Haddad. Challenges faced while simultaneously implementing cmmi and scrum: A case study in the tax preparation software industry. In *2012 Ninth Intl. Conf. on Information Technology - New Generations*, pages 314–318, 2012.
- [53] V. C. Nguyen and X. Qafmolla. Agile development of platform independent model in model driven architecture. In *2010 Third Intl. Conf. on Information and Computing*, volume 2, pages 344–347, 2010.
- [54] P. A. O. Salo. Agile methods in european embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum. *IET Software*, 2:58–64(6), 2008.
- [55] A. Oboler and I. Sommerville. Research documentation guidelines - capturing knowledge, improving research. In *Fourth Intl. Conf. on Information Technology (ITNG'07)*, 2007.
- [56] M. Ormsby and C. Busby-Earle. A standardized procedure to conceptualizing and completing user stories. In *2017 Intl. Conf. on Computational Science and Computational Intelligence (CSCI)*, pages 934–939, 2017.
- [57] J. Pasukmit, P. Thongtanunam, and S. Karunasekera. Towards just-enough documentation for agile effort estimation: What information should be documented? In *2021 IEEE Intl. Conf. on Software Maintenance and Evolution (ICSME)*, pages 114–125, 2021.
- [58] R. Phalnikar, V. Deshpande, and S. Joshi. Applying agile principles for distributed software development. In *2009 Intl. Conf. on Advanced Computer Control*, pages 535–539, 2009.
- [59] L. T. Portela and G. Borrego. Scrumconix: Agile and documented method to agsd. In *2016 IEEE 11th Intl. Conf. on Global Software Engineering (ICGSE)*, pages 195–196, 2016.
- [60] C. R. Prause and Z. Durdik. Architectural design and documentation: Waste in agile development? In *2012 Intl. Conf. on Software and System Process (ICSSP)*, pages 130–134, 2012.
- [61] A. Qusef. Test-to-code traceability: Why and how? In *2013 IEEE Jordan Conf. on Applied Electrical Engineering and Computing Technologies (AEECT)*, pages 1–8, 2013.
- [62] I. K. Raharjana, D. Siahaan, and C. Fatichah. User story extraction from online news for software requirements elicitation: A conceptual model. In *2019 16th Intl. Joint Conf. on Computer Science and Software Engineering (JCSSE)*, pages 342–347, 2019.
- [63] B. Ramesh, L. Cao, and R. Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Info Systems Journal*, 20(5):449–480, 2010.
- [64] A. S. B. Rani and A. R. N. B. Kamal. Text mining to concept mining: Leads feature location in software system. In *2018 IEEE Intl. Conf. on Computational Intelligence and Computing Research (ICCIC)*, 2018.
- [65] A. Read and R. O. Briggs. The many lives of an agile story: Design processes, design products, and understandings in a large-scale agile development project. In *2012 45th Hawaii Intl. Conf. on System Sciences*, pages 5319–5328, 2012.
- [66] S. G. Rojas and J. M. C. Mora. Source code documentation simul loco. In *7th Iberian Conf. on Info Systems and Tech (CISTI 2012)*, 2012.
- [67] E. Rubin and H. Rubin. Supporting agile software development through active documentation. *Requirements Engineering*, 16(2), Jun 2011.
- [68] S. Saito, Y. Iimura, A. K. Massey, and A. I. Antón. Discovering undocumented knowledge through visualization of agile software development activities. *Requirements Engineering*, 23(3):381–399, 2018.
- [69] S. Saito, Y. Iimura, A. K. Massey, and A. I. Antón. How much undocumented knowledge is there in agile software development?: Case study on industrial project using issue tracking system and version control system. In *2017 IEEE 25th Intl. Requirements Engineering Conf. (RE)*, pages 194–203, 2017.
- [70] B. Selic. Agile documentation, anyone? *IEEE Software*, 26(6), 2009.
- [71] M. Shafiq and U. s. Waheed. Documentation in agile development a comparative analysis. In *2018 IEEE 21st Intl. Multi-Topic Conf. (INMIC)*, pages 1–8, 2018.
- [72] A. K. Sharma, V. Deep, and N. Garg. An efficient way of articulation or suppression in agile methodologies. In *Confluence 2013: The Next Generation Information Technology Summit (4th Intl. Conf.)*, 2013.
- [73] C. Silveira, J. P. Faria, A. Aguiar, and R. Vidal. Wiki based requirements documentation of generic software products. In *Australian Workshop on Requirements Engineering (AWRE)*, 2005.
- [74] H. F. Soares, N. S. Alves, T. S. Mendes, M. Mendonça, and R. O. Spínola. Investigating the link between user stories and documentation debt on software projects. In *2015 12th Intl. Conf. on Information Technology - New Generations*, pages 385–390, 2015.
- [75] S. M. Sohan, F. Maurer, C. Anslow, and M. P. Robillard. A study of the effectiveness of usage examples in rest api documentation. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 53–61, 2017.
- [76] I. Sommerville. Software documentation. *Software engineering*, 2, 2001.
- [77] C. J. Stettina, W. Heijstek, and T. E. Fægri. Documentation work in agile teams: The role of documentation formalism in achieving a sustainable practice. In *2012 Agile Conf.*, pages 31–40, 2012.
- [78] C. J. Stettina and E. Kroon. Is there an agile handover? an empirical study of documentation and project handover practices across agile software teams. In *2013 Intl. Conf. on Engineering, Technology and Innovation (ICE) IEEE Intl. Technology Management Conf.*, 2013.
- [79] V. Surwase. Rest api modeling languages-a developer’s perspective. *Int. J. Sci. Technol. Eng.*, 2(10):634–637, 2016.
- [80] S. Voigt, D. Hüttemann, and A. Gohr. sprintdoc: Concept for an agile documentation tool. In *2016 11th Iberian Conf. on Information Systems and Technologies (CISTI)*, pages 1–6, 2016.
- [81] S. Voigt, J. von Garrel, J. Müller, and D. Wirth. A study of documentation in agile software projects. In *Proceedings of the 10th ACM/IEEE Intl. Symposium on Empirical Software Engineering and Measurement, ESEM '16*, 2016.

- [82] M. K. Wadiwala and A. F. M. Ishaq. Use of design and modeling in agile software development. In *2010 Second Intl. Conf. on Engineering System Management and Applications*, pages 1–5, 2010.
- [83] R. Witte, B. Sateli, N. Khamis, and J. Rilling. Intelligent software development environments: Integrating natural language processing with the eclipse platform. In C. Butz and P. Lingras, editors, *Advances in Artificial Intelligence*, pages 408–419, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [84] C. Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th Intl. Conf. on evaluation and assessment in software engineering*, 2014.