International Journal of Core Engineering & Management
Volume-6, Issue-12, 2021 ISSN No: 2348-9510

THE ROLE OF REQUIREMENT GATHERING IN AGILE SOFTWARE DEVELOPMENT: STRATEGIES FOR SUCCESS AND CHALLENGES

Rajesh Goyal
FS – Insurance
IBM - US
Email: rajesh.nim@gmail.com

Abstract

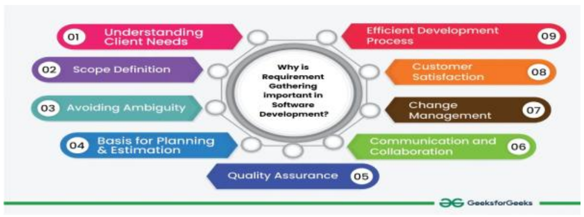
Agile techniques for software requirements engineering allow for prompt adjustments to take into account modifications to the client's software requirements. Agile requirements engineering approaches can, however, present some difficulties that impede the rapid and sustainable creation of software. In understanding the place of requirements gathering in Agile software development, this work concentrates on the procedural aspect of software development amidst change. Unfortunately, during the requirements gathering phase there are numerous problems ranging from incomplete, or conflicting, or even changing requirements which may inevitably lead to failure. The discussion uses examples such as the unsuccessful project at Qantas and thus these considerations give users the responsibility of providing input. Open-source software development is then explored in relation to requirements gathering and its associated difficulties. The paper also identifies other major Agile methodologies that provide a framework for requirement gathering. Various strategies for success in Agile requirements gathering, such as user stories, backlog refinement, collaboration, and prototyping, are discussed. The paper also addresses the challenges, including scope creep, ambiguity, and communication issues, proposing solutions to mitigate these risks. Agile principles and practices, when effectively applied, can ensure that requirement gathering leads to successful project outcomes, improved customer satisfaction, and innovative solutions.

Keywords: Requirements Gathering, Agile Software Development, Strategies and Challenges

I. INTRODUCTION

The software development world is abuzz with the agile method. As a response to the "need for an alternative to documentation-driven, heavyweight software development processes," agile methodologies emerged as a way to build software. Traditional approaches include gathering and documenting a "complete" set of requirements as the first step, then moving on to architectural and high-level design, development, and inspection. Some practitioners, starting in the 1990s, found these early stages of growth to be very difficult, if not impossible, to complete. Industry and technology are evolving at breakneck speeds, requirements "change at rates that swamp traditional methods," and consumers are becoming more and more ambivalent about upfront requirements while simultaneously demanding more from software. Therefore, several consultants have taken it upon themselves to adapt to the unavoidable shift by developing their own methodologies and practices. In reality, Agile methods are more of a collection of practices that adhere to common principles and share a common set of values [1].

Many, if not the vast majority, of the Fortune 500 organisations throughout the globe are beginning to embrace the agile software development (ASD) methodology, which is widely recognised as a best practice in the business sector. ASD is not a good approach to use for every kind of software development project, however, and it does not always provide good results. The present research outlines the essential Agile Software Development success variables that are associated with project success.
The goal of modern agile software development approaches, such Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Agile Modelling (AM), and others, is to create and deliver high-quality software that can satisfy all of the needs of the client. However, the majority of developer's neglect to take into account non-functional criteria like dependability, usability, scalability, and performance [2].
A nonfunctional requirement describes the features, attributes, and limitations of the program, while a functional requirement specifies the product's correct functioning. Because of the limited time, developers only focused on meeting the functional requirements. NFRs have the same responsibility as FRs in the development of software. The software may fail because the agile approach does not handle non-functional requirements. Software is produced effectively when NFR and FR are integrated [3]. Figure 1 shows the software requirement gathering.


Why is Requirement Gathering important in Software Development?
01 Understanding Client Needs
02 Scope Definition
03 Avoiding Ambiguity
04 Basis for Planning & Estimation
05 Quality Assurance
06 Communication and Collaboration
07 Change Management
08 Customer Satisfaction
09 Efficient Development Process
GeeksforGeeks
Fig. 1. Requirement Gathering software

Capturing, evaluating, and prioritizing needs in an iterative and collaborative way is the process known as agile requirements gathering. Agile stress constant input, adaptation, and flexibility throughout the project lifetime, in contrast to conventional waterfall methodologies. With this method, the demands of the stakeholders are satisfied and the product or solution is guaranteed to enhance the company. The paper contributes as:

- Emphasizes the critical role of requirements gathering in Agile for delivering quality software under changing conditions.
- Provides real-world examples, such as the Qantas project failure, to illustrate the consequences of poor requirements gathering.
- Explores the unique difficulties in gathering requirements for open-source software development.
- Outlines key Agile frameworks like Scrum, XP, and FDD for structured requirement elicitation.
- Discusses effective techniques, including user stories, backlog refinement, and prototyping, to enhance Agile requirements gathering.

1.1 Structure of the paper

The following paper are structured as: Section II provide the overview of requirements gathering, Section III give the overview of Agile Software Development, Section IV and V discussed the strategies for success in agile requirement gathering and challenges in agile requirement gathering. And last section VI provides the conclusion of this work with future directions.

II. OVERVIEW OF REQUIREMENTS GATHERING

Software development requires a complex method for acquiring requirements. Two significant obstacles in requirements collecting that may potentially contribute to the failure of software projects are incomplete requirements and requirements that are constantly changing. As a concrete illustration, consider the cancellation of a $40 million software project by Australia's main airline, Qantas, due to resistance from prospective customers, namely the Union of Aircraft mechanics. The fact that upper management was so set on implementing their own ideas on the new software without consulting or considering user feedback was a major problem. This seems to be a possible issue with open source software development as well, as OSS developers sometimes just state the requirements. Developers often disregard needs information provided by OSS users (such as feature requests made via issue reporting artefacts). This is partly because requirements data provided via issue reporting artefacts often includes issues, such as invalidity and incompleteness, which exacerbate other issues, such the inability to address bugs and the implementation of feature requests [4].

2.1 Issues and Solution for Requirement Gathering

Gathering requirements is an essential component of every project. It will become a difficult endeavor to accomplish. The knowledge expert's role is to compile the specifications. There are a few typical issues that arise while collecting requirements. In the same way that there is always a method to solve an issue, there are always suggestions on how to overcome a challenge while collecting requirements. A successful software project is guaranteed by high-quality requirements.

- **Undocumented Processes:** Many organizations either have inadequate or nonexistent documentation on their current practices. In this case, obtaining requirements becomes a two-step procedure. Firstly, the informational domain of the current process, followed by the identification of domains for enhancement and optimization.
- **Conflicting Requirements:** The business analyst will be responsible for the comprehensive documentation of all requirements. The ideas of all stakeholders will never be identical, which results in conflicting requirements. The business analyst recognizes requests that are inconsistent and allows stakeholders to determine their priorities.
- **Lack of Access to End Users**: End users were sometimes too preoccupied with their everyday tasks to take part in the demand collecting process. The absence of end users necessitates suitable resolution for a number of reasons.
- **Validating and Tracing Requirement**: Before beginning the implementation, the requirements that have been obtained and specified should be double verified. The needless necessity is avoided. The key will be to trace the requirements. Forward and backward traceability need to be present.
- **Stakeholder Design:** Rather of giving specifics on what the system should accomplish, stakeholders or end users need to express how the system should operate. Speaking with stakeholders about potential solutions may be insightful, but it can also refocus attention on real issues and lead to better solution ideas.

- **Communication Issues:** This includes terminology errors, ambiguous presumptions, and linguistic differences that cause miscommunication between stakeholders and business analysts. The easiest method to get around this is to set up two-way communication [5].

Requirement collecting is a continuous, flexible process in agile software development that is focused on stakeholder participation. Instead of comprehensive upfront documentation, requirements are collected and refined throughout the development cycle in the form of user stories—short, user-focused descriptions of features. Teams may respond to shifting demands and provide value gradually using this iterative method, which guarantees that the product changes in response to ongoing input from customers and the market.

III. AGILE SOFTWARE DEVELOPMENT

Agile development methodologies were created to address the challenge of providing high-quality software within a business context and needs that are changing quickly and continuously. The software and IT sectors have a track record of success with agile approaches.

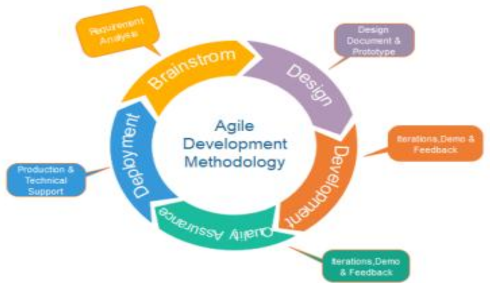

Requirement Analysis
Brainstrom
Design
Design Document & Prototype
Agile Development Methodology
Development
Iterations,Demo & Feedback
Quality Assurance
Iterations,Demo & Feedback
Deployment
Production & Technical Support

Fig. 2. Agile software development Methodologies[6]

Collaboration among teams and quick feedback from customers and other stakeholders are key to agile software development [7]. Sharing code, information, and concepts is what is meant by collaboration and communication. Social media hosted in the cloud is used for communication and collaboration. In addition, several methods and resources are offered for teamwork. These approaches to communication and teamwork are equally effective in on-site and off-site development environments. The team uses Skype for scrum meetings, wikis and discussion forums for team communication, and an AWS-EC2 instance for database sharing. Real-time reports are also used.

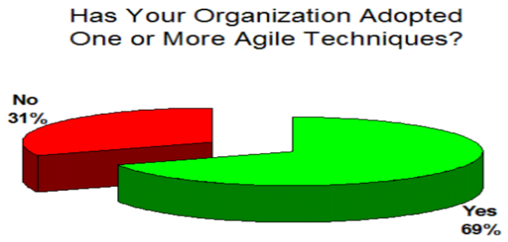


Fig. 3. Adoption of Agile Development

Figure 3 demonstrates that around 69% of organizations are implementing one or more agile methods for use in organizational development and general project management.

3.1Various approaches towards the development of Agile Software

- **Agile Modelling (AM):** is an innovative method for doing out modelling tasks. Developers may follow these steps, which are based on agile philosophy, to create models that address design issues and aid in documentation without "over-building" them. The goal is to minimize the number of models and supporting materials. Although some of the AM methods, including brainstorming, assist certain RE procedures, the RE techniques are not mentioned specifically in AM.
- **Feature-Driven Development (FDD):** is a simple, five-step approach that emphasizes generating a features list, designing and constructing phases with entrance and exit criteria, planning by feature, and then iteratively designing and building steps by feature. Developers and domain specialists work together in the first stage to build the generic domain model. The classes, connections, methods, and properties shown in the class diagrams form the backbone of the model.
- **Dynamic Systems Development Method (DSDM):** sprang up in the middle of the '90s in the UK. It developed from and is an extension of the RAD methodology. The DSDM process begins with two studies: the feasibility and the business. The initial needs are gathered over these two stages.
- **Extreme Programming (XP):** principles of openness, honesty, criticism, and bravery form its foundation. Software requirements might be unclear or subject to frequent change, but XP tries to make it possible to design effective software anyhow. A key component of the XP is the coordination and arrangement of the separate practices.
- **Scrum**: is a method based on hard data that prioritizes efficiency, adaptability, and versatility. Scrum gives developers a lot of leeway to pick and select which software development methodologies, strategies, and approaches to use throughout the implementation phase. Agile project management with an emphasis on 30-day sprints to achieve a defined set of backlog features is what Scrum is all about. Daily 15-minute team meetings for integration and coordination are the key practice of Scrum.

3.2 Principles of Agile Software Development:

Rather of being rule-based, agile approaches are principle-based and have established norms for interactions, responsibilities, and tasks. The software engineers on the team, together with the project manager, are guided by a set of principles that include [8][9]:

- The top focus is ensuring customer satisfaction via the timely and consistent supply of important software.
- Even in the latter stages of development, agile procedures are adaptable to new needs because they use change to the customer's benefit.
- Deliver functional software on a regular basis, preferably within a couple of weeks but not more than a couple of months.
- Businesspeople and developers must be involved in order for them to work together every day throughout the project.
- Collaborate on initiatives with enthusiastic people. Have faith in their abilities and provide them the environment and resources they need to succeed.
- When possible, communicate with members of a development team face-to-face to provide the most efficient and effective transfer of knowledge.
- The main indicator of progress is functional software.

3.3 Benefits of Agile Requirements Gathering Beyond Software Development

- **Improved Project Success Rates**: Through an iterative process that prioritises the delivery of incremental value and the consideration of stakeholder input, agile requirements gathering lowers the likelihood of project failures.
- **Enhanced Customer Satisfaction:** Agile guarantees greater customer happiness and loyalty by including them into the requirements collecting process and providing value early and consistently.
- **Increased Innovation:** Agile promotes a mindset of constant learning, feedback-driven enhancements, and experimentation, all of which contribute to an innovative culture. This drives innovation, which in turn drives corporate growth by meeting market wants.

IV. STRATEGIES FOR SUCCESS IN AGILE REQUIREMENT GATHERING

Agile software development emphasizes flexibility, collaboration, and iterative progress. However, effective requirement gathering, also known as requirement elicitation, remains a key success factor for Agile projects. The following success factors Agile Requirement Gathering discussed below:

1. User Stories and Epics

In Agile, requirements are normally defined in the form of user stories which are brief, basic statements of what user's desire from the system. These are alongside epics which are larger pieces of work that can be further divided into several user stories. It allows the team to break this down into user stories so that tasks are prioritized according to customer value while remaining as flexible as the requirements process requires.

2. Regular Backlog Refinement

The visible work product is kept in the product backlog that contains all of the desired features and the changes. In grooming the backlog daily, the user stories are always ready, prioritized and aligned to the goal of the project as set. This allows for controlling of scope and evolution of the requirements throughout the next cycles based on customer feedback.

3. Collaboration and Cross-functional Teams

Agile is based on teamwork with particular emphasis on the development team; the product owner and stakeholders and finally the user. Communication in Agile needs to be done face to face often when it comes to requirement gathering. Stakeholder participation gives the team a clear understanding of new needs to be met, and modifications may be effected as necessary.

4. Prototyping and Iterative Feedback

Generating early attempt or actual mock-up exposes layouts to stakeholders for discussion before committing hardware and software resource. This feedback loop ensures that requirements are user-friendly and reduces the occurrence of unwanted functionality by iteratively improving them.

5. Customer Involvement and Continuous Feedback

Customer interaction is a significant aspect of Agile, as Agile projects give much attention to customers. Customers or stakeholders through daily scrum meetings and at the end of each sprint like in sprint review or demo participate in the monitoring of progress hence real time feedbacks to the product backlog.

6. Acceptance Criteria and Definition of Done
Criterion of a user story acceptance makes it clear among the team of developers regarding when a particular feature is considered to be done. Thus, simplifying the concept through the Definition of Done (DoD) weakens possible confusion and boosts the accountability.

7. Prioritization Techniques (MoSCoW, Kano, etc.)
Scrum teams therefore always apply the MoSCoW (Must have, Should have, Could have, Won't have) and the Kano Models in ascertaining which of the requirements is most valuable to the user, with the implication that such priorities are solved first.

V. CHALLENGES IN AGILE REQUIREMENT GATHERING

Both, gathering and managing requirements in this environment is more of a challenging task and requires more of specific approaches and techniques. Some of the challenges discussed below:
1. **Changing Requirements and Scope Creep**
In most cases, anything labeled as an agile plan is flexible, and, unfortunately, flexible means that people tend to begin adding more features and functionality unbeknownst to the rest of your team. This is one of the strong sides of Agile, yet it can be difficult to manage a constant change without getting off track.

2. **Ambiguity and Incomplete Requirements**
In Agile, requirements are frequently in fact vague and not fully specified at first in order to undergo major changes in the future. This could be useful for increasing creativity and latitude but can result in ambiguity, and hence confusion to the development team, or to achieve the user's needs may take several cycles.

3. **Difficulty in Prioritizing Features**
In Agile, what features should be developed first must always be in dialogue between the stakeholders, product owners, and developers. This is because there is conflict of priorities where the investors may want the company to work on other features which are not so valuable in the market yet the company focuses on delivering those features which its clients highly value.

4. Balancing Long-term Vision with Short-term Delivery
In Agile, development is incremental, but it is challenging to manage long-term objectives alongside daily expectations for the arrival of new increments. One common set-back is that teams tend to concentrate on the near-terms goals and forget the importance of big pictures.

5. **Misaligned Stakeholder Expectations**
Cross-functional Agile teams can have numerous and quite divergent requirements of the process from a variety of stakeholders. Otherwise these expectations may become dissatisfactions, particularly if entities do not understand the Principles of Agile and revert to conventional strategies, with established dates and rigid perimeters.

6. **Inadequate Documentation**
It is a weakness that Agile pays more attention to producing tangible developed software instead of documenting all angles of the project early on to enhance efficiency in the later phases. It would also be as a nightmare to onboard new members to the team or review the older features that will need a little tweaking here and there among other issues caused by a lack of documentation.

7. **Team and Stakeholder Commitment**
Agile requires continuous involvement from stakeholders, but this is not always possible. When stakeholders are unavailable or unresponsive, the team may struggle to gather accurate requirements and feedback, affecting progress.

8. **Communication in Distributed Teams**
Agile works best with face-to-face interactions, but with the rise of remote work and distributed teams, real-time communication may be harder to achieve. This can lead to misunderstandings or misaligned goals in requirement gathering.

VI. LITERATURE REVIEW

A literature overview on the topic of demand collecting in agile software development, including successful solutions and problems, is presented in this section.
This De Lucia and Qusef, (2003) paper explores issues related to requirements engineering in agile software development processes and offers solutions to some of the problems that arise from using agile requirements engineering practices on large projects. Some of these problems include dealing with sensitive requirements, both functional and non-functional, keeping agile teams connected with external customers, and documenting and managing requirements documentation [10].
In this de Haan and Cohen, (2007) research presents findings from a scenario of an entrepreneurial off-the-shelf software manufacturer in numerous case studies. A novel approach to software development was born out of these discoveries. Lead-driven development involves creative vendors improvising software development procedures to correct bugs and add new features depending on leads they get from one marketing meeting or demonstration to the next. This kind of development caters demo features to the demands of prospective consumers, but it doesn't help the product's commercial development in the long term [11].
In this Kumar and Singh, (2016) research, the writers addressed the function of Agile software development methodology in creating software for public art installations. This article aims to talk about software engineering problems and how the software development model's agility affects the end product [12].
The Shafiq and Waheed, (2018) research on documentation, its function, process model documentation templates, pitfalls to avoid, data collection for documentation, document requirement format, and principles for process structure documentation—including XP, lean, crystal, FDD, DSDM, MSF, AUP, and ASD—in the context of documented processes. Researchers and practitioners alike may use the survey results to better understand which process model is most suited to a given scenario [13].
This Tshabalala and Khoza, (2019) the effect of a conflict-risk that is well handled in Agile teams on the technological competitiveness of the company is examined in this article. The quantitative approach to research, which used a purposive sample strategy, was the methodology that was

used. A survey that was conducted online to obtain information from Agile experts around South Africa was used in the research. The results show that conflict exists and is very important for accomplishing the objectives of agile projects. Furthermore, an organization's capacity to sustain effective technological competitiveness, which contributes to the organization's overall performance, is impacted by successful conflict-risk reduction and management, according to the results [14].

This Wang et al. (2014) study investigates eight agile groups from four software development organizations, presenting a survey with three research questions to evaluate the handling of RE in actual agile development. In order to address the three study topics, we created a questionnaire specifically for 108 participants with extensive expertise in agile development. Although many agile approaches promote coding without waiting for written requirements and design specifications, our study reveals that agile RE practices are a vital precondition for projects' success and play a significant role in agile development [15].

This Fitriani, Rahayu and Sensuse, (2016) study's overarching goal is to draw attention to the difficulty of ASD by conducting a comprehensive literature review. The inclusion criteria were met by a total of 20 publications in the sample. This research presents the distribution of publications according to specified parameters, such as years and nations. The findings revealed that there are thirty obstacles to using ASD methodology. Out of the 30 issues associated with ASD, the most critical ones are team management and dispersed teams. Then there are requirements prioritization, documentation, altering and over scoping, organization, process, and ongoing feedback and progress monitoring [16].

The following table 1 provides the summary of the related work for requirement gathering in agile software development.

TABLE I. SUMMARY OF RELATED WORK

Reference	Methodology	Key Findings	Challenges Identified	Suggestions for Improvement
[10]	Analysis of RE activities in large agile projects.	Discusses handling sensitive and non-functional requirements in large projects.	- Poor management of sensitive and non-functional requirements - Issues with documentation and customer communication	Suggestions include better documentation practices and stronger customer communication.
[11]	Multiple case study of entrepreneurial off-the-shelf software vendors.	Emergence of lead-driven development, where vendors improvise software features based on customer demos.	- Short-term focus on demo features - Lack of long-term product development planning	Highlights the risks of overemphasizing customer-specific demo features at the expense of long-term product goals.
[12]	Case study on agile development for art installations.	Examines software engineering and agility impacts on art installation software.	- No specific challenges related to requirement gathering mentioned	General discussion of agility's role, but no improvement suggestions specific to RE.
[13]	Survey on documentation practices across different agile	Provides insights into document creation and requirements documentation in various	- Documentation challenges in methods like XP, DSDM, and others	Offers guidance on process model selection based on specific documentation needs.

	methodologies.	agile models.		
[14]	Quantitative survey on conflict-risk management in agile teams.	Conflict management plays a crucial role in agile project success and organizational competitiveness.	- Conflict within agile teams that can affect project goals	Encourages effective conflict-risk mitigation strategies for better project outcomes.
[15]	Survey with 108 agile professionals from four organizations.	Agile RE practices are critical to project success despite coding often beginning without formal requirements.	- Over-reliance on coding without formal requirements - RE not formalized in many agile methods	RE should be recognized as a vital component and formalized in agile methods for project success.
[16]	Systematic literature review of 20 papers.	Identifies 30 challenges in agile software development.	- Team management - Distributed team coordination - Requirement prioritization -Documentation - Scope changes - Monitoring and feedback issues	Highlights a need for improved team management, documentation practices, and process monitoring in agile settings.

VII. CONCLUSION

The foundation of successful Agile software development is effective requirements gathering. Although Agile enables teams to adjust to evolving customer demands, it also presents obstacles such as controlling scope creep, settling competing objectives, and guaranteeing adequate documentation. Agile methodologies such as Scrum and XP offer frameworks to address these issues through iterative feedback, cross-functional collaboration, and continuous customer involvement. Overcoming challenges like ambiguity, stakeholder misalignment, and distributed team communication requires focused strategies, including regular backlog refinement, prototyping, and the use of prioritization techniques. Ultimately, Agile requirements gathering, when managed correctly, fosters innovation, customer satisfaction, and project success by ensuring that software is developed in close alignment with evolving user needs.
Future work in Agile requirements gathering should focus on enhancing tools for remote collaboration, exploring AI-driven techniques for prioritizing user requirements, and integrating user experience (UX) methodologies within Agile frameworks. Additionally, empirical studies on the effectiveness of various requirements gathering strategies across different Agile methodologies will help refine best practices and improve project outcomes.

REFERENCES

1. A. De Lucia and A. Qusef, "Requirements Engineering in Agile Software Development," J. Emerg. Technol. Web Intell., vol. 2, no. 3, Aug. 2003, doi: 10.4304/jetwi.2.3.212-220.
2. C. . Kavitha and S. M. Thomas, "Requirement Gathering for small Projects using Agile Methods," IJCA Spec. Issue Comput. Sci. - New Dimens. Perspect., 2011.
3. T. Suryawanshi and G. Rao, "A Survey to Support NFRs in Agile Software Development Process," Trupti Suryawanshi al, / Int. J. Comput. Sci. Inf. Technol., vol. 6, no. 6, pp. 5487–5489, 2015.

4. S. Lane, P. O'Raghallaigh, and D. Sammon, "Requirements gathering: the journey," J. Decis. Syst., 2016, doi: 10.1080/12460125.2016.1187390.
5. J. Mushtaq, "Different Requirements Gathering Techniques and Issues," Int. J. Sci. Eng. Res., vol. 7, no. 9, pp. 835–840, 2016.
6. G. S. Matharu, A. Mishra, H. Singh, and P. Upadhyay, "Empirical Study of Agile Software Development Methodologies," ACM SIGSOFT Softw. Eng. Notes, vol. 40, no. 1, pp. 1–6, Feb. 2015, doi: 10.1145/2693208.2693233.
7. D. Batra, W. Xia, and M. Mingyu ", "Collaboration in Agile Software Development: Concept and Dimensions," Commun. Assoc. Inf. Syst., vol. 41, pp. 429–449, 2017, doi: 10.17705/1CAIS.04120.
8. T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," Inf. Softw. Technol., vol. 50, no. 9–10, pp. 833–859, Aug. 2008, doi: 10.1016/j.infsof.2008.01.006.
9. H. Alahyari, R. Berntsson Svensson, and T. Gorschek, "A study of value in agile software development organizations," J. Syst. Softw., vol. 125, pp. 271–288, Mar. 2017, doi: 10.1016/j.jss.2016.12.007.
10. A. De Lucia and A. Qusef, "Requirements engineering in agile software development," J. Emerg. Technol. Web Intell., 2010, doi: 10.4304/jetwi.2.3.212-220.
11. U. de Haan and S. Cohen, "The Role of Improvisation in Off-the-Shelf Software Development of Entrepreneurial Vendors," in 2007 International Conference on Systems Engineering and Modeling, IEEE, Mar. 2007, pp. 85–92. doi: 10.1109/ICSEM.2007.373337.
12. A. Kumar and Y. Singh, "The impact of agile based software engineering in interactive art installations," in 2016 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA) (Fall), IEEE, Sep. 2016, pp. 1–5. doi: 10.1109/ICACCAF.2016.7749009.
13. M. Shafiq and U. sman Waheed, "Documentation in Agile Development A Comparative Analysis," in 2018 IEEE 21st International Multi-Topic Conference (INMIC), IEEE, Nov. 2018, pp. 1–8. doi: 10.1109/INMIC.2018.8595625.
14. M. M. Tshabalala and L. T. Khoza, "Improving the Organization Technology competitiveness through Effective Management of Conflict-Risk within Agile Teams," in Proceedings - 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC 2019, 2019. doi: 10.1109/IMITEC45504.2019.9015899.
15. X. Wang, L. Zhao, Y. Wang, and J. Sun, "The Role of Requirements Engineering Practices in Agile Development: An Empirical Study," in Communications in Computer and Information Science, 2014, pp. 195–209. doi: 10.1007/978-3-662-43610-3_15.
16. W. R. Fitriani, P. Rahayu, and D. I. Sensuse, "Challenges in agile software development: A systematic literature review," in 2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS), IEEE, Oct. 2016, pp. 155–164. doi: 10.1109/ICACSIS.2016.7872736.