Implicit Thinking Knowledge Injection Framework for Agile Requirements Engineering

Kaiss Elghariani[1], Nazri Kama[2], Nurulhuda Firdaus Mohd Azmi[3], Nur Azaliah Abu bakar[4]
Razak Faculty of Technology and Informatics
Universiti Teknologi Malaysia
Kuala Lumpur, Malaysia

***Abstract*—Agile has become commonly used as a software development methodology and its success depends on face-to-face communication of software developers and the faster software product delivery. Implicit thinking knowledge has considered as a very significant for organization self-learning. The main goal of paying attention to managing the implicit thinking knowledge is to retrieve valuable information of how the software is developed. However, requirements documentation is a challenging task for Agile software engineers. The current Agile requirements documentation does not incorporate the implicit thinking knowledge with the values it intends to achieve in the software project. This research addresses this issue and introduce a framework assists to inject the implicit thinking knowledge in Agile requirements engineering. An experiment used a survey questionnaire and case study of real project implemented for the framework evaluation. The results show that the framework enables software engineers to share and document their implicit thinking knowledge during Agile requirements documentation.**

Keywords*—Software development methodology; agile methodology; requirements engineering; requirements documentation; and implicit thinking documentation

I. INTRODUCTION

The implicit thinking knowledge has become very significant issue for researchers. The number of researches have been increased since that the implicit thinking knowledge considered as a primary key of success through the ability of team members to create and share their implicit thinking knowledge in software product. This consideration has attracted researchers in investigating the ways in which implicit thinking knowledge can be successfully captured, identified, categorized, shared, and documented. The topic of documenting the implicit thinking knowledge has arisen to address this need. Thus, a few knowledge frameworks designed to define practices related to knowledge. Consequently, in Agile methodology, software requirements are defined and documented in the form of user story cards. Besides, as one of Agile features that a clear discussing about user requirements is conducting by having a regular meeting [14], for example, in Scrum approach every day meeting is conducted to have to check the requirements implementation for maintaining the iterations. Traditional software development methodologies are focus more on documenting all software development phases while Agile methodology does not provide comprehensive documentation. Knowledge is easily and explicitly captured in traditional software development methodologies while the Agile methodology deals with implicit thinking knowledge. The implicit thinking knowledge is kept in the software developers' minds [17]. The most critical part of capturing the implicit thinking knowledge is to retrieve the implicit thinking knowledge from their minds, so as to incorporate the right knowledge at the right requirement when needed and to encourage innovation.

Agile methodology has few inherent practices that assist sharing experiences software developers' knowledge during developing the software, for instance, some of Agile practices helps the process to overcome the challenges of capturing implicit thinking knowledge such as pair programming, face-to-face communication and [12]. The frequent interaction among Agile software developers provides an environment that supports the implicit thinking knowledge sharing and cooperative knowledge detection. In addition, implicit thinking knowledge might be feast more efficiently by documenting the face-to-face communication than and stored in databases to be retrieved by knowledge workers [3].

The mentioned practices help to manage knowledge. Also, other approaches for capturing implicit thinking knowledge requires more consideration. This paper proposes a framework injects the implicit thinking knowledge of software engineers in Agile requirements documentation (IITKARD).

A quantitative method such as survey questionnaire has been used in a case study of a real software project for a purpose of the framework evaluation. During the experiment, software requirements used in a real software project converted into user Agile user story cards as a form of Agile requirements engineering. In order to improve the efficiency and usability of findings, a survey questionnaire has been conducted to collect data from focus group of Agile software engineering experts. Therefore the results of the survey have to evaluate the efficiency and the usability of the proposed framework and its usefulness in Agile software methodology.

The proposed implicit thinking knowledge injection provides a systematic process for Agile software engineering during the requirements engineering phase to inject their implicit thinking knowledge in Agile requirements documentation. The framework will provide a systematic process to successfully address the issue of neglecting of implicit thinking knowledge in Agile requirements engineering. This paper divided into five sections, firstly section 2, which discusses the implicit thinking knowledge and Agile requirements engineering, followed by section 3, which describes the proposed IITKARD framework. Then section 4

discusses the results of the evaluation experiment and finally the conclusion.

II. Implicit Thinking Knowledge and Agile Requirements Engineering

The implicit thinking knowledge management includes a set of processes that control the creation, interaction, and sharing of knowledge [14]. It is a few steps used during developing the software to extract and share developers' knowledge to assist them understand how the software is built. The retrieved knowledge needs to be organized so that they are clear and searchable [5]. Moreover, managing the implicit thinking knowledge also includes strategies to create an environment of sharing knowledge and supportive tools that assist the process of injection of software developers' thinking knowledge, in turn, to learn from each other.

Implicit thinking knowledge management is not only a technical issue [1]. Since it relies on people expertise, the success of managing the implicit thinking knowledge is highly influenced by human. However, these factors named a non-technical factors, which are related to the characterization of human implicit thinking knowledge, and the promoting the knowledge workers to contribute to share their implicit thinking knowledge in order to build shared knowledge repository [3].

Unfortunately, knowledge workers are unenthusiastic to share their own implicit thinking knowledge with other people [5]. Some workers consider their knowledge as a private professional knowledge and they are not often intending to share it [15]. The workers reluctance of sharing knowledge might also be contributed by other organizational practices. The staff, who are rewarded due to their expertise might lead to that knowledge might be kept in their minds [5]. Moreover, psychological issues might also effect the participation of the knowledge workers [16]. For example, in the organizations meetings some workers try to avoid to share their knowledge and this would adversely affect the meetings process as a result [19]. According to Hissen [1] some members are hardly to speak, due to their low status, shyness or controversial ideas. Therefore, the externalization of implicit thinking knowledge should be part of a defined knowledge and it should not be left an optional. For example, in software engineering field, software engineers have deliberations includes different views of software projects lifecycle reviews should be captured. Thus, more attention need to paid regarding to the importance of the integration between software development methodologies and knowledge management.

Obviously, that implicit and explicit are different in terms due to implicit thinking is hard to document. Sandra et al. [20] highlighted that implicit thinking knowledge documentation is not something for discussion sense [21]. Though, the significance of capturing implicit thinking knowledge during requirements engineering led the researchers to pay more attention on topics related implicit thinking knowledge documentation [18]. However, tacit and implicit knowledge are slightly different [22]. Researches stated that implicit thinking knowledge is not organized knowledge but it can be structured and documented. On the other hand, organizing the implicit thinking knowledge is not easy. Moreover, implicit thinking knowledge categorized as an expression, assumptions, developers thought that, might be translated to principle. According to Correia and Aguiar [8] there is insufficient information of implicit knowledge, while [9] argues that implicit thinking knowledge is "how" to do things, but some issues to be explicitly described.

Additionally, the technical issues contribute in the difficulties of capturing the implicit thinking knowledge, the implicit thinking knowledge is not well structures and there is not any standard format to be followed. Therefore, using traditional data models could hardly capture he implicit thinking knowledge. Few models in the software engineering aimed to codify the implicit thinking knowledge such as DRL [4], QOC [11] and IBIS [13]. Even though, these models support the implicit thinking knowledge capturing, but it is hardly to cover all implicit thinking knowledge. Sometimes the tacit knowledge is differently expressed; it might be a body language expression. Thus, these all variations of implicit thinking knowledge expression is hardly to be accommodated.

Elghariani and Kama [7] stated that Agile requirements engineering practices assist to resolve the traditional requirements engineering challenges and highlighting appropriate capability of teams' cross-functional development [2]. In addition, in software industry, some challenges have been faced during practicing Agile requirements engineering such as less documentation and ignoring non-functional requirements documentation e.g. usability and security [7]. Neglecting of implicit thinking knowledge documentation is considered as a major issue for both Agile and traditional software methodologies [10]. One of Agile requirements engineering practices is creating user story cards, which includes few attributes related to the user software requirement, for example, story card number, story date, story priority, story description, etc. [6]. Therefore, Shim and Lee [25] addressed that ignoring implicit thinking knowledge documentation in Agile methodology cause some major issues as following:

- Time waste of asking the same questions by software developers.
- Software issues might be frequently faced by developers but they forget how were solved.
- Losing information once particular developer left the project.
- Lack of recording developer's communication.
- Software usability issue
- Neglecting of unstructured knowledge contribution.

The major challenges of documenting the implicit thinking knowledge in Agile requirements engineering is how implicit information software engineers can be converted into explicit information, as well as how to convert explicit information from individual software engineer to groups in the organization [21]. The retrieved knowledge needs to be recorder and stored in a repository. Additionally, the process of sharing software engineer's implicit knowledge needs to be followed by all software developers [15]. Also, this includes understanding the

purpose of storing and documenting the implicit thinking knowledge [12].

Chau and Maurer [23] stated that there are serious issues to capture team members' implicit thinking. Most of the members are not able to exactly define their thinking. So Ahmed [24] suggested that software developers need to have more face-to-face discussion on software requirements. Another suggestion is using electronic databases will solve the issue, if the face-to face communication is not possible then some alternatives can be used such as e-mail, social media, audio and video conferencing [18].

III. Related Work

Based on the previous studies, few frameworks were proposed to manage the knowledge using traditional software development methodologies. A framework for supporting knowledge management (KMS) recommends two significant considerations, which are, the existence and structure of the knowledge [12].

The proposed framework KMS layered into seven layers known as interface, access, collaborative, application, transport, integration and repositories [22]. In addition, Hahn and Subramani [12] proposed an approach for knowledge management system known as Soft-System Knowledge Management. This approach was proposed to confirm the suitability between the organizational requirements on new product development (NPD) and knowledge management creativities. There are three main components are included in this framework, which are Knowledge sharing methods, Organizational level and Key Enablers.

A Knowledge Management System in Open Source Software Development Environment (KMSOS2oD) framework is particularly designed to support Open Source Systems development life cycle and it includes five core components, which are layers, components, process, knowledge, and Communities of Practice (CoP) [15].

Several tools have been proposed by researchers to support distributed Agile software development, and those tools are especially developed to support sharing applications. The proposed tools adopted the concept of collaboration and communication among team members. Therefore, the tools used are messengers, e-mails and newsgroups. Tools like videos/audios conferencing are also used as a real-time collaboration tools [8]. Moreover, tools providing a Plug-in like JSPWiki and MASE are used to support environment tailored for Agile development teams [17].

Kavitha and Irfan [17] have proposed a framework to capture knowledge retrieved from the Agile team members, which may either be distributed or collocated. The framework helps to increase the organizational learning process to capture and maintain the knowledge on the required information. The framework included a set of integrated tools that support capturing and storing knowledge. The tacit knowledge sharing was through a real-time collaboration tools such as news groups, NetMeeting and e-mails.

Shim and Lee [25] suggested an approach to capture and manage knowledge. This approach was built by using the main characteristics of the social software and expands them to merge the knowledge and its structure then manage them probably by using online tools. The authors used web technologies to extract the implicit thinking knowledge and injected to the knowledge base; the injecting was used with the platform as provider to enable the users to retrieve a documented knowledge related to the certain software practice, which also affect the user interaction.

Based on literature related to knowledge sharing within software development, there are many approaches aimed at this objective. Most of the proposed frameworks aimed for managing software knowledge designed for traditional methodologies [17]. Some approaches are realized practically, while others are simply theoretical such as Rayan and O'Connor [22] that propose a theoretical model for defining, acquiring and sharing tacit knowledge surfaced through social interaction. KMSOS framework is another approach, which is introduced by Lakulu [15], but it is mainly targeting capturing software knowledge within the Open source software process. However, it does not cater for documenting any sort of implicit thinking knowledge. In terms of approaches targeting the Agile software methodology, though [17] and [25] are most related to our work as they both targeting software knowledge management for Agile software methodology. Nonetheless, both are aimed to tackle the knowledge sharing in all phases of the Agile lifecycle. We on the hand, only focusing on the requirement elicitation phase. We believe that the requirements for capturing implicit thinking knowledge vary in different Agile phases. Hence, a generic knowledge-capturing model would not cater for the capturing richness of the requirements engineering phase. Because it is well known that the requirements engineering phase accommodates most of the deliberations made by the Agile team. However, though both [17] and [25], are targeting the capturing of implicit thinking knowledge within the Agile software process, but they ignored the importance of the User Story Cards, which as at the core of implicit thinking knowledge generation. The same approaches ([17], [25]), also lacks the formal representation of the implicit knowledge, they instead adopting a social media tools as repository to store the implicit thinking knowledge.

IV. The Proposed Knowledge Framework

The framework is proposed to be part of reality [1]. It is meant to summarize the conceptual framework of the reality being demonstrated. Essentially, our proposed framework is built to inject the implicit thinking knowledge created while practicing Agile software requirements documentation. The proposed framework facilitates sharing implicit knowledge [5].

The Internet or local network usually connects Agile software team members. The framework supported by a tool that assists in collecting the knowledge. Experiences generated in the form of arguments categorized as issue, assumption, suggestion, question and opinion are entered by the software engineer and stored in the knowledge repository.

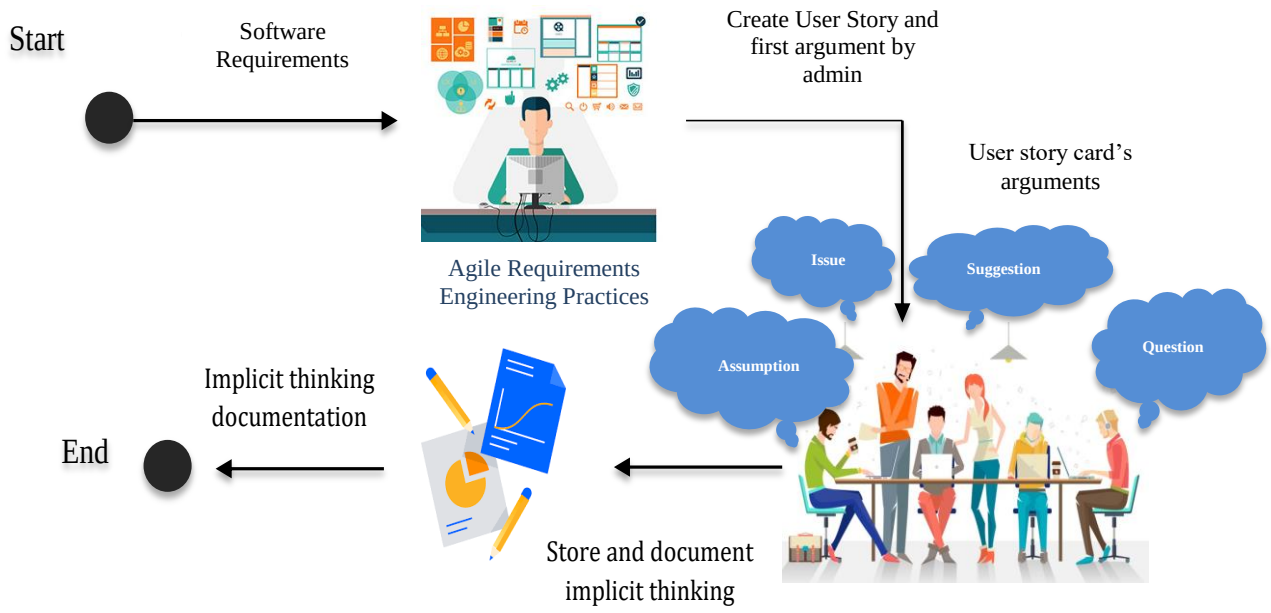

Fig. 1. Conceptual View of IITKARD Framework.

Fig 1 shows a conceptual view of the proposed framework. It shows the stages of the processes of sharing software engineering implicit thinking knowledge in Agile requirements engineering. Based on the proposed IITKAD framework, the knowledge generation loop starts from creating the user story card, which will be the root of the experience knowledge.

The IITKARD framework helps to share informal knowledge. The implicit thinking knowledge generated through the IITKARD framework helps to maintain the up-datedness of the captured knowledge. Moreover, this framework provides Agile software engineers with a supportive tool that helps in injecting the implicit thinking knowledge. The user story card created by the admin (team leader) and adding the first argument of the retrieved knowledge related to requirements engineering practices are entered via an interface by the admin (team leader) and stored in the knowledge database. The implicit knowledge extracted during developing the software requirements is accessible to all team members and easy to find. Agile team members will be able to retrieve the information related to certain user story card from database as structured information. Efficient query mechanisms also provided, which can be helpful to retrieve the required data.

The proposed IIKARD framework, which has four main activities, which are: (1) cerate the user stories; (2) set team leader first arguments; (3) inject the implicit thinking knowledge of software engineers; (4) document the implicit thinking knowledge. Therefore, to evaluate the framework efficiency, a tool was developed to systematize the processes of developing the framework as shown in Fig 2. The tool of the proposed framework has four main steps, which are:

Step 1 Create User Story: The first practice of Agile requirements engineering is to create the user story of each requirement and the proposed framework tool requires the team leader to key in the user story information for example: Story Title, Task Engineer and describe the story (requirement).

Step 2 Set Team Leader first Argument of the user story: The second step of the proposed framework tool is setting the first argument by the team leader (Admin).

Step 3 Inject the implicit thinking knowledge of software engineers: The Third step of the proposed framework tool is to let the team members to key in arguments such as Issues, Assumptions, Suggestions, Questions and Opinions.

Step 4: Document the implicit thinking knowledge: The final step of the proposed framework tool is to document the arguments of each user story by displaying the arguments in sort of storyline. It contains the title and its arguments member's photo, name, text and icon of the argument type whether it is an issue, assumption or suggestion. Hence, this, IITKARD framework was successfully designed and developed to achieve this research objective in order to provide a framework to inject the implicit thinking of software engineers in Agile RE.

Therefore, IITKARD framework controls the deliberation among team members and supports their arguments by detecting the type of the arguments and simplifies the communication about certain issue. The tool which implements IITKARD framework steps shall be applied and helpful for areal data from a real software project adopting Agile methodology as a software development methodology.

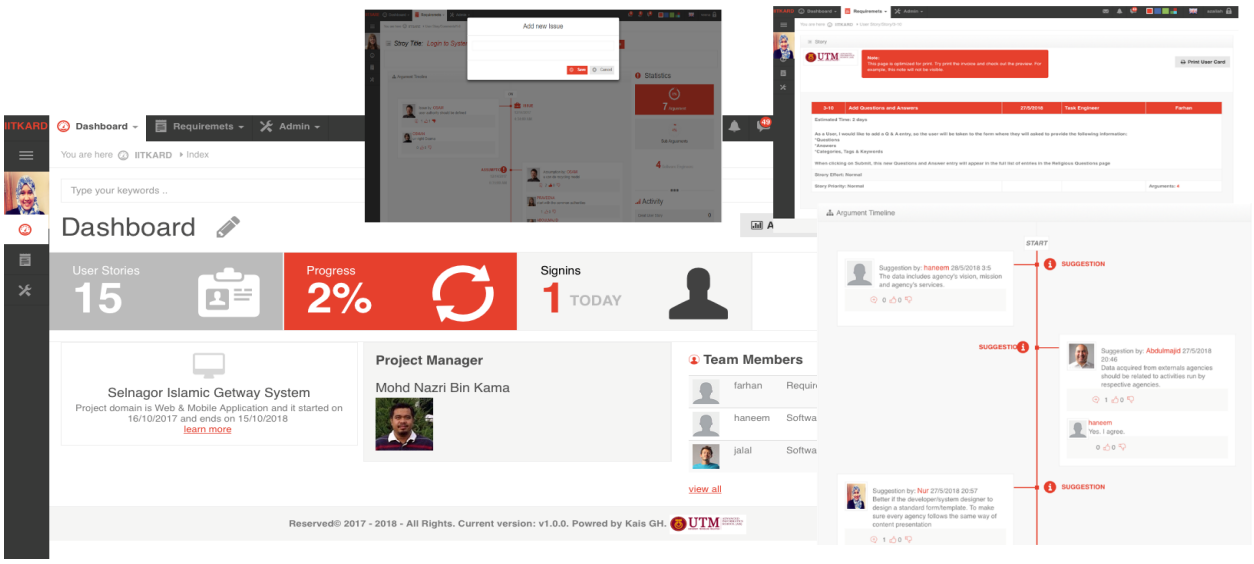

Fig. 2. An Implementation of the Proposed Framework.

V. Results

After performing, the first experiment, evaluation survey questionnaire has been distributed to focus group of experts. The 13 completed questionnaires have been received from 10 Agile software engineering experts. The experts came from different organizations with the experience of Agile software methodology and requirements engineering. The participants started performing the experiment by using traditional Agile requirements documentation. However, the experiment shows that the documentation does not include any implicit thinking knowledge in the user story cards. The experiment shows that the overall results obtained a high level of experts' satisfaction in terms of IITKARD efficiency evaluation. In general, the overall results shows that the participants performed as an experts' in Agile requirements engineering and improves that IITKARD is more efficient in Agile methodology since Agile lean on a strong communication among the team members.

The traditional requirements documentation used for the experiment does not include any implicit thinking knowledge shared by software engineers during developing software system which been adopted as a real project data for this experiment. The experts' participants in the experiment shows their satisfaction on the efficiency of IITKARD due to the improvement of ITKARD framework that can assist software engineers who are newly involved in the software project to understand how the requirement was developed. Also, the experiment proved that the IITKARD improves the usability of injecting the implicit thinking knowledge in Agile requirements documentation. It is noted that the improvement rate of the IITKARD usability is high in Agile methodology.

The proposed models of knowledge's management in both traditional and Agile methodology [8][17][18][19][25], use a social media tools to support sharing and recording the knowledge, but it is well known that social media knowledge is not structured enough, and any searching through this knowledge would not be efficient in terms of retrievability. IITKARD on the other hand, proposed a custom knowledge model, which strikes a balance between ease of use and retrievability, as shown through the tool developed to demonstrate the process of injecting the implicit thinking knowledge among the team members. This is achieved by managing the requirements (User Story Cards) and the implicit knowledge. Therefore, the experts' perceptions of the outcome show that the overall results of the experiment are at satisfactory level in terms of the usability of the framework.

VI. Conclusion

Software engineering is a domain consists a very knowledge thorough and realized in implicit form in many software companies. Knowledge management systems were designed more for structured knowledge while implicit thinking knowledge is ignored. The implicit thinking knowledge is usually held in professionals' minds that might be lost at any time. Therefore, there is significant need to exploit implicit thinking knowledge as well. This paper proposes a framework that injects implicit thinking knowledge in Agile requirements engineering. The framework integrates both explicit knowledge in the form of user story cards and implicit thinking knowledge in the form of arguments that compose the context of the creation criteria of the knowledge captured. In order to inject the implicit thinking knowledge in Agile requirements engineering the captured knowledge documented in the form of user story arguments.

VII. Limitation and Future Work

Even though the research contributions, a few limitations in this research are considered. The designed IITKARD framework has suggested that the shared implicit thinking knowledge should be modified and deleted. The injecting process suggestion was made to the software requirements engineering practices. There are two elements exist in the software requirements engineering process that are dependent to one and another. The requirements engineering elements are Requirements analysis, Requirements Documentation. This research study applied only for requirements engineering as the input and output of IITKARD framework and this is one of the limitations of this study. Therefore, it is important to apply

IITKARD framework to all Agile methodology phases since all the shared knowledge will cover all the phases, which could affect the overall software development process.

Based on the limitations described above, the following areas may constitute possible future work:

- Extending the research study by covering a bigger and more complex Agile software projects as samples of the case selections. Big and complex Agile software projects may involve a longer software requirements implementation and more team members who can interact with proposed IITKARD framework.
- The software requirements engineering elements consist of analysis, documentation in each of the software requirement. These two main Agile requirements engineering elements are correspond to one and another. For example, if one requirement is deleted or changed in the software engineer should consider the implicit thinking knowledge related to that particular requirement to be modified or deleted as well. In this study, the proposed IITKARD framework concentrated only on the Agile requirements documentation. The suggested future work could extend this study by implementing the IITKARD framework all Agile methodology phases.
- Since this study has used academic context to run the experimentation, a future work can replicate the experiment design in the industry context to examine the practitioners' perceptions in using the IITKARD framework. In addition, it can be extended whereby the evaluation does not only measure the practitioners' perceptions but it can also measure the software engineering experts satisfaction of the end product that is developed using the framework of injecting the implicit thinking knowledge in Agile requirements engineering.

ACKNOWLEDGMENT

The author would like to express the acknowledgment to Razak Faculty of Technology & Informatics for the support given in conducting this research study.

REFERENCES

[1] Abdulmajid Hissen, "Capturing Software-Engineering Tacit Knowledge", Departement of Computer Science, Sebha University, 2008 2nd Eroupean Computing Conference.

[2] Aneesa Rida Asghar, Shahid Nazir Bhatti, Atika Tabassum and S Asim Ali Shah, "The Impact of Analytical Assessment of Requirements Prioritization Models: An Empirical Study". International Journal of Advanced Computer Science and Applications, Vol. 8, No. 2, 2017

[3] Bjarnason, E., Wnuk, K., & Regnell, B. (2011b). Requirements are slipping through the gaps—A case study on causes & effects of communication gaps in large-scale software development. In 2011 IEEE 19th international requirements engineering conference (pp. 37–46).

[4] Claudia Müller, Julian Bahrs, Norbert Gronau, "Considering the Knowledge Factor in Agile Software Development", Journal of Universal Knowledge Management, vol. 0, no. 2 (2005), pp.128-147

[5] David Walsh Palmieri, "Knowledge Management through Pair Programming", Thesis

[6] Deki Satria, Dana Indra Sensuse and Handrie Noprisson, 2017 "A systematic literature review of the improved Agile software development", in 2017 International Conference on Information Technology Systems and Innovation (ICITSI). Bandung, Indonesia, PP 94 – 99

[7] Elghariani, K., and Kama, 2016. Review on Agile requirements engineering challenges,in 3rd International Conference on Computer and Information Siences, Kuala Lumpur, Malaysia, pp. 507-512.

[8] Filipe Figueiredo Correia, Ademar Aguiar, ". Software Knowledge Capture and Acquisition: Tool Support for Agile Settings", IEEE Computer, 2009

[9] Meira Levy, Orit Hazzan, "Knowledge Management in Practice:The Case of Agile Software Development", IEEE software, 2009, pp. 60-65

[10] Ghani I., Azham Z., and Jeong SR. "Integrating Software Security into Agile-Scrum Method". KSII Transactions on Internet an Information Systems (TIIS) 8 (2), 646-663. 2014.

[11] Gerardo Canfora, Aniello Cimitile, Corrado Aaron Visaggio, "Lessons learned about distributed pair programming: what are the knowledge needs to address?" IEEE Software, 2003

[12] Hahn, J., & Subramani, M. ''A Framework of Knowledge Management System: Issues and Challenges For Theory and Practice'', Proceedings of the twenty first international conference on Information systems, ACM, Brisbane, 2000

[13] Harald Holz and Frank Maurer, "Knowledge Management Support for Distributed Agile Software Processes", 2003

[14] Inayat et al., (2014). A systematic literature review on Agile requirements engineering practices and challenges. Computers in Human Behavior. (2014).

[15] Modi Lakulu, "A Framework of Collaborative Knowledge Management System in Open Source Software Development Environment" Journal of computer and Information Science, Vol.3, No.1, Feb 2010, pp.81-90

[16] Mustafa Ally, Fiona Darroch, and Mark Toleman, "A Framework for Understanding the Factors Influencing Pair Programming Success", XP 2005, Sheffield, UK, pp.18–23

[17] R. K. Kavitha and M. S. Irfan Ahmed, 2011. "A Knowledge Management Framework for Agile Software Developme nt Teams", in 2011 International Conference on Process Automation, Control and Computing. Coimbatore, India, PP 1-5

[18] Saeed S. and Alsamdi I., 2013. "A software development process for open source and open competition projects", International Journal of Business Information Systems, 12(1), 110-122

[19] Sandra L., Buitrun Francisco J., Pino, Brenda L., Flores-Rios, Jorge E., Ibarra-Esquer, Marua Angulica Astorga-Vargas, 2017. "A Model for Enhancing Tacit Knowledge Flow in Non-functional Requirements Elicitatio", in 2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT). Mérida, Mexico

[20] Shankar, R., Acharia, S., & Baveja, A. ''Soft-system Knowledge Management Framework for New Product Development'', Journal of Knowledge Management, 2009, 13(1) , pp.135-153.

[21] Sufian Fannoun ; John Kerins, 2017. "Evaluating current practice and proposing a system to enhance knowledge assets within a small software development unit", 2018 4th International Conference on Information

[22] Ryan, S. and O'Connor, R., Acquiring and Sharing Tacit Knowledge in Software Development Teams: An Empirical Study, Information and Software Technology, Vol. 55, No. 9, pp. 1614 - 1624, 2013.

[23] Thomas Chau, Frank Maurer, "Integrated Process Support and Lightweight Knowledge Sharing for Agile Software Organizations", 2005

[24] Usman Ahmed, 2018. "A review on knowledge management in requirements engineering", in 2018 International Conference on Engineering and Emerging Technologies (ICEET), Lahore, Pakistan, PP 1-5.

[25] Woogon Shim and Seok-Won Lee, 2017. "An Agile Approach for Managing Requirements to Improve Learning and Adaptability", in 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW). Lisbon, Portuga, PP 435-438