Security Benefits for Agile Software Development

S. Hassan Adelyar
Institute of Informatics,
Tallinn University,
Tallinn, Estonia
adelyar@tlu.ee

Alex Norta
Department of Informatics,
Tallinn University of Technology,
Tallinn, Estonia
alex.norta.phd@ieee.org

***Abstract*-Agile methodologies such as scrum and Extreme Programming (XP) are efficient development processes by accepting changes at any phase and delivering software quickly to customers. However, these methodologies have been criticized because of the unavailability of security as an important quality goal of software systems. Although there are pre-existing research results on this topic, there is no pure approach for identifying security benefits of agile practices that relate to the core "embrace-changes" principle of agile. Specifically, we analyze agile practices to find the security benefits in customer- and developer activities. Identifying these benefits supports the secure development of software using an agile methodology.**

Keywords-Agile; Development-process; Embrace-changes; Software-security; Security-benefits; Security-principles

I. INTRODUCTION

The aim of this paper is to analyze agile practices in order to identify security benefits during software-development processes throughout the customer- and developer activities. Agile is an iterative and incremental software development approach and each iteration involves the team to go through a full development cycle [1]. The focus of agile is on developers and customers with the objective to produce working software quickly [2], [3], [4]. Today, many software-development organizations are using agile software development because agile produces faster and more cost-effective software solutions while maintaining a high rate of customer satisfaction [5], [6], [7], [8], [9], [10], [3]. However, agile methodologies such as extreme programming (XP) and scrum do not pay attention to security features because the working software and iterative delivery are the primary measure of success [9], [11]. At the same time, considering the current attacking landscape, security is an important non-functional requirement of software products.

Since software security is a quality aspect, therefore it is important to think about security at each stage of software development. Agile practices are carried out by developers and it is necessary to consider security issues during the development process throughout the customer- and developer activities. According to [12], software is vulnerable to threats that may occur during software-development processes and inadequate practices of software development can lead to insecure software [13].

One way to incorporate security into a development team is by identifying security benefits of agile practices. These benefits improve security of software by incorporating security principles [14] into agile features such as customer- and developer interaction, short iterations and responses to changes. Applying and incorporating security principles into agile processes from the early stages of software development, supports developers to adopt agile methodologies for secure software development [15], [16]. Therefore, our aim is to analyze agile practices in order to identify security benefits based on the security principles defined in [14]. Security principles are a criteria for measuring and identifying security benefits in developer- and customer activities. Experiences of practitioners show that security principles guide the design and implementation of software without security flaws.

This paper demonstrates how to improve agile methodologies for producing secure software by considering the security benefits of agile. This is accomplished by analyzing agile practices to identify what activities of customers and developers are most beneficial for secure agile software development. This paper is a continuation of our previous work in which we analyzed agile practices in order to identify security challenges based on the security principles [17]. The results of this paper assist software developers to understand where to integrate security measures and which software development phases are important in order to develop secure software.

We conduct case-study based research [18] about the development process of applications that follows agile practices to analyze the relationship between security principles [14] and security benefits of agile practices. For data collection, we use both interviews and focus-group methods. During the case study, our special attention is to identify what activities of customers and developers are most compatible and beneficial for secure agile software development.

The rest of the paper is organized as follows. Section II provides a summary of existing literature. Section III contains additional information relevant for agile practices, software security and security principles. Section IV presents a brief overview of our case-study approach. In Section V, we present the results of our research. Finally, Section VI concludes this paper by summarizing the research work, giving the contributions achieved and showing directions for future work.

II. RELATED WORK

There exist many publications that criticize agile methodology because of unavailability of security elements in its development phases. On the other hand, in response to the increasing rate of security issues caused by security

vulnerabilities in software products, many researchers publish about security integration with agile practices [19], [20] and [21]. Researchers also found that many of the agile practices comply for building secure software [22], [23], [24]. These works and publications aim at adopting agile practices to secure software development.

A group of researchers study agile practices and discuss its potential benefits for secure software development [25], [26], [27], [28], [11]. Other researchers study agile methodology for integrating security into a specific practice, such as refactoring, in order to take advantage of these practices for secure software development [29], [30].

However, based on literature, we found a gap pertaining to a holistic approach for identifying security benefits in agile practices based on a set of security principles [14]. Applying security principles at the early stage of software development is a better solution for producing secure software [16], [15]. Since agile focuses on communication, self-organization and the collaboration between developers and customers, therefore applying security principles on developer- and customer activities helps the developer team to understand security concern. Our approach also aims at increasing the security knowledge of developers by considering security related parts in agile practices.

III. BACKGROUND

In this section we briefly explain the concepts that are useful in understanding our approach. Agile practices, described in Section III-A, software security, described in Section III-B and security principles, described in Section III-C, are important elements in our research.

A. Agile Software Development

Agile is a dominant approach for software development and it is based on the concept of agility. In general, agility is the ability to provide effective response to change, communication among team members and delivery of working software in short duration. Agile methods such as extreme programming, scrum and adaptive software development are all based on a set of general principles that are defined by the agile alliance and manifesto of agile software development [31].

The cornerstone of agile methodologies is the practices that help to produce software quickly. The twelve practices of agile are: planning-game, on-site customer, metaphor, small-releases, simple-design, pair-programming, collective-ownership, coding standards, 40-hour-week, continuous-integration, refactoring and testing.

For our research, we categorize these practices into three phases: the main practice in the first phase is the planning-game practice. Four other practices, indirectly involved, are on-site customer, metaphor, simple-design and small-release. The on-site customer practice is to involve the customer for writing and prioritizing user stories. Small-releases and simple design practices means it is up to the customer of the software to make important decisions. The second phase includes the practices to implement the user stories and the main practice in this phase is pair programming in which two programmers are coding together. Other practices involved in this phase are coding-standards, simple-design, small-releases, collective-ownership and 40-hour-weeks. In the last phase, the implemented features in the current iteration are integrated to the software and continuous integration is the main agile practice in this phase. Pertaining to the simple-design and refactoring practices, the developers constantly redesign and refactor relevant parts of the system. The testing practice of agile is to achieve the desired quality of the software.

B. Software Security

Security is a quality aspect of a system property that reflects the ability to protect itself from accidental or deliberate attacks. Security is a composite of the attributes confidentiality, integrity, availability and accountability [32], [33]. Confidentiality is defined as the prevention of unauthorized exposure of software code and execution. Integrity is the preventions of software code and execution from unauthorized alterations, amendment or deletion. Availability is the ability of software to be available when needed, executed in a predictable way and delivers results in a predictable time frame. Accountability is the availability and integrity of the identity of the person who performs an operation.

C. Security Principles

Security principles are defined by [14] and guide a software design and implementation without security flaws. These principles are concepts or guidance that can be followed to develop secure systems during a software development stage. Applying security principles on the software development process also helps non-security expert developers to understand security concerns. The following is the list of security principles [14]:

Separation of Privileges: To develop secure software, the development process needs to verify the identity of developers and customers based on their privileges and responsibilities.

Least Common Mechanism: Minimize the amount of mechanism common to more than one user. That means customer- and developer activities in each practice of agile should be controlled separately.

Least Privileges: Every program and every user of a system should operate using the least set of privileges necessary to complete a job.

Complete Mediation: Every access to every object must be checked for authority.

Fail-safe Defaults: The default situation is lack of access, and the protection scheme identifies conditions under which access is permitted.

Economy of Mechanism: Keep the design as easy, simple and small as possible.

Psychological Acceptability: Design the human interfaces for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

Open Design: The design should not be secret and the mechanisms should not depend on the ignorance of potential attackers.

In the following section we describe our case study approach. Our case study consists of: case study design,

validation procedures, data collection procedure and data analysis procedures.

IV. CASE STUDY DESIGN

We choose a case-study based research method [18] and our aim is to identify security benefits in developer- and customer activities of agile practices based on the security principles as listed in Section III-C. Our aim is refined into the following research questions: How to identify agile security benefits during changes to software? To establish a separation of concerns, the main research question is divided into the following sub questions: What are security benefits of response-to-changes based on security principles? What are the tasks that improve security benefits in agile software development? Which agile practices have more security benefits?

We answer these questions with our case-study data collection and analysis. The case for our study is a software development process using agile practices described in Section III-A [18]. We select three different software development teams in Kabul city for interviews and one group of six developers as a focus-group. The subject for our study is security benefits in agile practices. Our case study has a deductive nature and therefore, we pose a set of hypotheses for the research. The main goal for using hypotheses is to identify security benefits based on the security principles, therefore we derive the hypotheses from security principles.

According to the security principles of Section III-C, a secure system can be studied by assessing four main characteristics which are: separation, restriction, simplicity and awareness [34]. Separation supports the accountability attribute of security. Accountability requires a clear definition of roles and responsibilities for each team member. The principles of "Separation of Privileges" and "Least Common Mechanism" help separation. These two principles support the accountability attribute of security. Restriction supports the confidentiality attributes of security. The principles of "Least Privileges", "Complete Mediation" and "Fail-safe Default" help restriction. Based on these three principles each member of the development team should be given only enough privileges to perform their duties. Integrity requires validation of activities and system-wide view and controls. Simplicity assures that the development- team activities are valid and correct. The principles of "Economy of Mechanism", "Psychological Acceptance" and "Open Design" help simplicity. Therefore, these tree principles also support integrity of the development-team activities. Software- developer attention and awareness is required to supports confidentiality, integrity, availability and accountability.

The following hypotheses are inferred from security principles and they are related to the above mentioned characteristics of a secure system.

(i) Continuous changes-to-software renders the process of separation of privilege easier that support accountability.

(ii) Continuous changes-to-software makes it easier to control the system-wide view of the software to support confidentiality and availability.

(iii) Continuous changes-to-software help the simplicity of software, which supports integrity.

(iv) Continuous changes-to-software improve the developer attention and supports confidentiality, integrity, availability and accountability.

The above hypotheses provide a useful bridge between security attributes, security principles and the interview with focus-group questions. Each hypothesis has a specific and clear aim and the collected data will confirm or reject that aim. These hypotheses, derived from the security principles, guide the preparation of focus-group and interview questions for gathering data about agile security. The results of analysis, either confirm or reject the hypotheses, which leads to either confirmed or rejected theories about agile security [18].

A. Validity Procedures

For improving the data validity, we carefully design our study implementing the qualitative investigation measures and data validity rules in all phases of our case study. For ensuring credibility, we carefully infer hypotheses from security principles [14] and then we deduce the interview questions from the hypotheses. Since the direct questions about security are difficult to answer, we use security principles as a bridge between the knowledge level of the researchers and interviewees. During the interviews, for some questions, an iterative questioning method is used for establishing more clarity of the questions. The collected data we code in such a way that the most serious threats to data validity are avoided. During the analysis phase we take care to correctly generalize our findings.

B. Data Collection Procedures

For answering our research questions we use two direct data collection methods that are focus-group discussions and interviews. We consider a focus-group comprising six developers and conduct interviews with 10 software developers. All the interviewees and focus-group members use an agile software development methodology and each member of the team has at least experience from three software-development projects. The interview questions are derived from the hypotheses, listed in Section IV-A, which are ordered according to security principles listed in Section III-C. The same questions are asked for the three main phases of agile practices, planning-game, pair-programming and continuous-integration. The mentioned three phases are collaborative and the activities of developers and customers in these practices are interdependent. The interviews and focus-group discussions are audio recorded into WMA multimedia files.

C. Analysis procedure

The main goal of analysis is to understand whether theories about the security benefits in agile practices are valid by testing the hypotheses. To achieve this goal, we analyze the collected data with the following steps:

1) First we formulate a set of themes that group the related codes. Each theme belongs to a hypothesis.
2) We read all the texts in the collected data and mark where the codes fit into the themes.
3) Results of the coding are analyzed per theme and presented.

Table I shows our predefined themes and a brief description from which a corresponding theme is derived.

Table I: Themes and Themes description

Theme	Theme Description
Separation of privileges	To see how continuous changes to software make the process of separation of privilege easier to implement.
Restriction of privileges	Restrict privileges, check every access and deny access during mistakes.
Software simplicity	Make the design of software simple, small and easy.
Improve attention	To see how continuous changes to software improve the developer attention.

The above themes are derived from our research hypotheses and deduced from security principles that relate to our research questions. During software development, if the activities of developers and customers are in compliance with the security principles, then it reduces security flaws and vulnerabilities in the software.

Table II and Table III show the coding results for the focus-group discussions and interviews respectively. We use a simple formula to evaluate what codes have more value for analysis. The formula is:

Code-value = Sources * Phases

In this formula, sources denote how many attenders in the focus-group or interview mention the code and phases denote the availability of code in the three main phases of agile software development. The possible values for phases are 1, 2, and 3.

Codes are sorted based on their value and then we review every theme separately and draw conclusions. We abbreviate the name of each phases where PG denotes planning-game, PP denotes pair-programming and CI denotes continuous-integration practice of agile. The value column gives the formula result.

Table II: Table of Focus-group Coding Results

Theme / Code	Sources	Phases	Value
1. Separation of Privileges			
Continuous integration and customer feedback eliminate future disputation.	8	PG+PP+CI	24
Incremental development and periodic customer feedback clarify the responsibilities of customer and developers.	7	PG+PP+CI	21
Iterative work increases developer ability to know the source of a problem and solve the problem effectively.	7	PG+PP+CI	21
Face-to-face interaction with a customer clarifies responsibilities.	5	PG+CI	10
Sharing of ideas among pairs limits errors and solve problem effectively and bring clearness in the developer responsibilities.	5	PP+CI	10
2. Restriction of Privileges			
Small increment and customer feedback maintain a system-wide view.	6	PG+PP+CI	18
Iterative work with customer presence increases our understanding for the software.	5	PG+PP+CI	15
With iterative work and customer explanation, we know all parts of the system.	4	PG+PP+CI	12
Working step by step improves developer control of the system.	4	PG+PP+CI	12
Changing programming pairs sustain the system-wide view and control.	5	PP+CI	10
Customer presence helps us to understand and control the software from the beginning.	2	PG+CI	4
3. Software Simplicity			
Working on one task at a time simplifies software development.	6	PG+PP+CI	18
Working on each part separately simplifies software development.	5	PG+PP+CI	15
The discussion of pairs simplifies software development.	6	PP+CI	12
Customer feedback simplifies software development.	6	PG+CI	12
In pair programming, ideas are shared, which increases simplicity.	5	PP+CI	10
Our previous work and experiences make the software development simpler.	3	PG+PP+CI	9
4. Attention & Awareness			
Iterative work increases our understanding and attention about the software.	5	PG+PP+CI	15
Working in pairs causes concentration and competition that result more attention.	6	PP+CI	12
On-time feedback of customer increases our attention about the development process.	6	PG+CI	12
Pairs' discussion increases developer attention.	5	PP+CI	10
Since the tasks are divided, therefore each one can concentrate on his work.	3	PG+PP+CI	9
Continuous integration & getting the desired result improve our attention.	5	CI	5
Customer can reject our work that increase our attention.	4	CI	4

Continuous Integration results customer feedback that increase our attention.	1	PG+PP+CI	3

Table III: Table of Interview Coding Results

Theme / Code	Sources	Phases	Value
1. Separation of Privileges			
By assigning privileges, each one know their responsibilities well.	6	PG+PP+CI	18
On-site customer prevents misuse of responsibility.	5	PG+PP+CI	15
Working step by step by customer with assigned privileges both customer and developer will focus well on their jobs.	5	PG+PP+CI	15
By assigning privileges each one tries to accomplish their tasks in the best way.	5	PG+PP	10
Each work gets integrated and finalized by approval and confirmation of both sides.	3	CI	3
2. Restriction of Privileges			
Customer feedback and pair programming reduce security gaps as one person will be verifier of codes.	6	PG+PP+CI	18
Continuous integration reduces unwanted changes to the software.	5	PG+PP+CI	15
Continuous customer feedback and integration minimize errors.	3	PG+CI	6
Incremental development makes changes easier.	3	CI	3
Working in pairs prevents misuse of privileges.	2	PP	2
3. Software Simplicity			
Determining the privileges, improve the quality of software.	6	PG+PP+CI	18
Feedback and idea of customer (from non-technical point) simplifies the software.	6	PG, CI	12
Different idea form pairs simplifies the software.	6	PP	6
Working on each part separately clarifies and simplifies the overall system.	4	CI	4
Incremental development makes changes easier.	3	CI	3
Pair programming reduces duplication that improves simplicity.	3	PP	3
Continuous integration removes errors that cause simplicity.	2	CI	2
4. Attention & Awareness			
Periodic feedback from customer increases developer attention.	5	PG+PP+CI	15
Iterative work increases developer focus on the software.	5	PG+PP+CI	15
Pair programming reduces possible errors and gaps.	5	PP	5
Continuous integration improves developer self-confidence.	4	CI	4
During the planning game, developers and customers solve problems and raised questions.	2	PG	2
Continuous integration improves knowledge about the software.	2	CI	2

For answering our research questions we analyze Table II and Table III to better make the coded data for identifying security benefits, the related tasks and agile practices that contain more benefits. Therefore, we combined Table II and Table III into Table IV, during combination we derived the benefits and related tasks from theme / code column of Table II and Table III. The practices column of Table IV is derived from the phase column of Table II and Table III. For each theme we select the three top ranked codes. The result of this process is shown in Table IV. We abbreviate the name of agile practices such as planning-game to PG, pair-programming to PP, continuous-integration to CI, on-site customer to OC, collective ownership to CO, refactoring to RF, small release to SR, simple design to SD, coding standards to CS, and metaphor to MP.

Table IV: Benefits, Tasks and Agile Practices

Benefits	Tasks	Practices
Separation of Privileges		
Elimination of future disputes	Incremental Development, Customer Feedback	PG, PP, CI, OC, CO, RF
Clarity of customer & developer responsibilities	Incremental Development, Customer Feedback	PG, PP, CI, OC, SR, RF
Identification of problem source	Iterative Work	PG, PP, CI, SR, CO, RF
Knowing of responsibilities	Privileges Assignment	PG, PP, CI, CS, RF
Prevention of misuse of responsibilities	Customer Feedback	PG, PP, CI, OC, CO, MP
Focus on work	Step-by-step Work	PG, PP, CI, SR, SD, CS, RF
Restriction of Privileges		

System-wide view	Small Increment, Customer Feedback	PG, PP, CI, OC, SR, SD, CS, RF
Understanding the software	Iterative Work	PG, PP, CI, SR, SD, CS, RF
Knowing all parts of the software	Iterative Work, Customer Feedback	PG, PP, CI, OC, SR, SD, CS, RF
Verification of code	Customer feedback & working in pairs	PG, PP, CI, OC, SR, SD, CS, RF
Prevention of unwanted changes	Incremental development	PG, PP, CI, SR, SD, CS, RF
Prevention of errors	Customer feedback, Incremental development	PG, CI, OC, MP, SR, CS, RF
Software Simplicity		
Simplicity of software development	Working on one task at a time	PG, PP, CI, SR, SD, CS, RF
Simplicity of software development	Working on each part separately	PG, PP, CI, SR, SD, CS, RF
Simplicity of software development	Discussion of pairs	PP, CI, SR, SD, CO, RF
Improvement of software quality	Privileges determination	PG, PP, CI, SR, SD, CS, RF
Simplify software	Customer feedback & ideas	PG, CI, OC, MP, SR, SD, CS, RF
Make changes easier	Incremental development	CI, CS, SR, SD, RF
Attention & Awareness		
Understanding & attention improvement	Iterative work	PG, PP, CI, SR, SD, CS, RF
Concentration & competition	Working in pairs	PP, CI, CS, CO
Improvement of developer attention	On-time customer feedback	PG, CI, OC, MP, RF
Improvement of attention	Customer Feedback	PG, PP, CI, OC, SR, CO, CS, RF
Improvement of developer focus on the software	Iterative Work	PG, PP, CI, SR, CS, RF
Reduction of errors	Working in Pairs	PP, SR, CO, CS, RF

V. RESULTS

As mentioned earlier, our goal is to analyze agile practices for identifying security benefits in customer- and developer activities based on security principles. For achieving this goal, we derive our hypotheses from security principles and then for each hypothesis, we determine a corresponding theme by collecting data via focus-group and interviews. Through the coding session, data collected from the focus-group and interview is organized into corresponding respective themes. From Table IV we can answer our research questions. We use excel sheet to sort the benefits and practices. After analyzing Table IV we found the following results:

Our First theme belongs to the security principles of "Separation of Privileges" and "Least Common Mechanisms". Based on these principles, the secure software development process must verify the identity of developers based on their responsibilities and minimize common mechanisms to more than one developer. From Table IV the benefits of clarity of customer & developer responsibilities, elimination of future disputes and identification of problem-source are the benefits to support this theme. All these benefits supports the "accountability attribute" of security, described in Section III-B. The tasks that improve these benefits are customer feedback, incremental development and privileges assignment. The most important practices for these benefits and tasks are planning game, pair programming, continuous integration, refactoring, on-site customer, small release and collective ownership.

Next, our second Theme belongs to "Least Privileges", "Complete Mediation" and "Fail-safe Default" security principles. Based on these principles, security attributes have a system-wide nature and the protection and authorization mechanisms for developing secure software, requires the restriction of privileges. From Table IV the benefits of system-wide view, understanding the software, knowing all parts of the software, verification of code, prevention of unwanted changes and prevention of errors are the benefits to support this theme. All these benefits supports the "confidentiality and availability attributes" of security, described in Section III-B. The tasks that improve these benefits are incremental development, customer feedback, iterative work and pair working. The most important practices for these benefits and tasks are planning game, continuous integration, refactoring, on-site customer, small release and collective ownership.

The third theme we derive from "Economy of Mechanism", "Open Design" and "Psychological Acceptability" security principles. Based on these principles, the design must be simple and small since techniques such as line-by-line inspection are necessary for finding security flaws in the code of software. For such techniques to be successful, a small and simple design is essential [14]. The focus-group and interviewees pointed to the simplicity of software development,

software- quality improvement, and ease of changes as the benefits that improve integrity. The tasks of working on one task at a time, working on each part separately, discussion of pairs, privileges determination, customer feedback and incremental development are tasks that improve the simplicity of software. The most important practices for these benefits and tasks are continuous integration, small release, simple design, refactoring, planning game and coding standards.

Finally, the last Theme of "Attention and Awareness", we derive from the security principle of "Fail-safe Defaults". This principle emphasizes security mechanisms that require high attention of developers during the whole software development process. From Table IV the benefits of understanding & attention improvement, concentration & competition, improvement of developer focus on the software and reduction of errors are the benefits to support this theme. These benefits supports the confidentiality, integrity, availability and accountability attributes of security, described in Section III-B. The tasks that improve these benefits are iterative work, working in pairs, and on-time customer feedback. The most important practices for these benefits and tasks are pair programming, continuous integration, coding standards, refactoring, planning game and small release.

VI. CONCLUSION AND FUTURE WORK

In this paper, we conduct a case study to identify and explain security benefits of agile software development by evaluating developer- and customer activities. Focus-group and interviews were used as a main source of evidence to collect data. An analysis of the collected data was performed to evaluate the relationship of security benefits and agile practices based on security principles [14]. The result of our study shows that adequate tasks and activities of developers- and customer, reduce security flaws and vulnerabilities from the developed software. Table V shows the tasks (customer- and developer activities), its security benefits and agile practices for security benefits.

Table V: Beneficial tasks, benefits and agile practices

Tasks	Benefits	Practices
Customer Feedback	Separation of privileges, Restriction of privileges, Software simplicity, Developer attention	PG, PP, CI, OC, CO, RF, SR, MP, SD, CS
Discussion of Pairs	Software simplicity	PG, CI, SR, SD, CO, RF
Incremental Development	Separation of privileges, Restriction of privileges, Software simplicity	PG, PP, CI, OC, SR, RF, SD, CS, CO, MP
Iterative Work	Separation of Privileges, Restriction of privileges, Developer attention	PG, PP, CI, SR, CO, RF, SD, CS
Privileges Assignment	Separation of privileges, Software simplicity	PG, PP, CI, CS, RF, SR, SD
Small Increment	Restriction of Privileges	PG, PP, CI, OC, SR, SD, CS, RF
Step-by-step Work	Separation of privileges	PG, PP, CI, SR, SD, CS, RF
Working in Pairs	Restriction of Privileges, Developer attention	PG, PP, CI, OC, SR, SD, CS, RF, CO
Working on each part separately	Software simplicity	PG, PP, CI, SR, SD, CS, RF
Working on one task at a time	Software simplicity	PG, PP, CI, SR, SD, CS, RF

Table V shows that many tasks in developer- and customer activities of agile methodologies improve software security through "Separation of privileges", "Restriction of privileges", "Software simplicity", and "Developer attention".

As a limitation of this research, the interviewed developers have little knowledge about software security and we are not able to design our interview- and focus-group questions to directly address software security. Instead, we derive the focus-group and interview questions based on the security principles [14] to address indirectly the security issues in software development process. The lower security knowledge and awareness of many software developers is also counted as a main source for security flaws during agile software development. Further studies and future work for introducing visual and easier methods well help to raise security awareness of agile software developers.

REFERENCES

[1] C. Lan and B. Ramesh, "Agile Requirements Engineering Practices: An Empirical Study," 2008.

[2] Sonia and A. Singhal, "Integration Analysis of Security Activities from the perspective of agility," *978-0-7695-4657-5/12 $26.00 © 2012 IEEE*, 2012.

[3] C. Pohl and Hans-JoachimHof, "Secure Scrum: Development of Secure Software with Scrum," *MuSe-Munich IT Security Research Group*, 2015.

[4] P. Abrahamsson, O. Salo, J. Ronkainen and J. Warsta, Agile Software Development Methods: Review and Analysis, Finland: VTT Electronics, 2002.

[5] H. John, "Agile Software Construction.," 2005.

[6] B. Gabrielle, "Rolling out agile in a large enterprise.," in *41st Hawaii International Conference on System Science.* , 2008.

[7] A. Sidky, J. Arthur and S. Bohner, "A Disciplined Approach to Adopting Agile Practices: The Agile Adoption Framework, Innovations in systems and software engineering.," 2007.

[8] I. Ghani and I. Yasin, "Software Security Engineering in Extreme programming Methodology: A Systematic Literature Review," *ISSN 1013-5316; CODEN: SINTE*, pp. 215-221, 2013.

[9] S. Bartsch, "Practitioners Perspectives on Security in Agile Development," *Sixth International Conference on Availability Reliability and Security*, pp. 479-484, 2011.

[10] B. Beca, "Agile Development with Security Engineering Activities," pp. 149-158, 2011.

[11] J. Wayrynen, M. Boden and G. Bostrom, "Security Engineering and eXtreme Programming: An Impossible Marriage".*Comminications Security Lab, Ericsson Research.*

[12] Cappelli, Dawn, Trzeciak and Randall, *Insider Threats in the SDLC. Presentation at SEPG*, 2006.

[13] C. Mann, "Why Software is so Bad?," in *Technology Review*, 2002.

[14] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," pp. 1278 - 1308, 1975.

[15] F. Eduardo B., "A Methodology for Secure Software Design," in *DBLP*, Florida, 2004.

[16] M. J. Peterson, J. B. Bowles and C. M. Eastman, "An approach for integrating security into UML class design," in *IEEE Southeast Conference*, 2006.

[17] H. Adelyar and A. Norta, "Towards a Secure Agile Software Development Process," pp. 101-106, 2016.

[18] P. Runeson, M. Host and A. Rainer, Case Study Research in Software Engineering, New Jersey, USA: John Wiley, 2012.

[19] M. Siponen, R. Baskerville and T. Kuivalainen, "Integrating security into agile development methods., 2005.," in *In Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005.

[20] V. Kongsli, "Towards Agile Security in Web Applications. In the Proceedings of OOPSLA," in *OOPSLA*, Oregon, USA, 2006.

[21] G. Bostrom and B. Konstantin, "Extending XP Practices to Support Security Requirements Engineering," in *ICSE*, University of British Columbia, Canada, 2006.

[22] P. Amey and R. Chapman, "Static Verification and Extreme Programming.," in *Proceedings of the ACM SIGAda Annual International Conference.*, 2003.

[23] O. Murro, R. Deias and G. Mugheddo, "Assessing XP at an European Internet Company.," 2003.

[24] J. Shore, "Continuous Design. IEEE Software, Vol. 21 (1).," 2004.

[25] B. Konstantin, "Extreme Security Engineering: On Employing XP Practices to Achieve Good Enouht Security," *First ACM Workshop on Business Driven Security Engineering*, p. 7, 2003.

[26] I. Ghani, N. Izzaty and A. Firdaus, "ROLE-BASED EXTREME PROGRAMMING (XP) FOR SECURE SOFTWARE DEVELOPMENT," *Special Issue-Agile Symposium*, pp. 1071-1074, 2013.

[27] A. Chandrabose and K. Alagarsamy, "Security Requirement Engineering - A Strategic Approach," *International Journal of Computer Applications*, vol. 13, pp. 25-32, 2011.

[28] C. Wood and G. Knox, "Guidelines for Agile Security Requirements Engineering".

[29] E. Aydal, R. Paige, H. Chivers and P. Brooke, "Security Planning and Refactoring in Extreme Programming," *Springer Link*, vol. 4044, pp. 154-163, 2006.

[30] Sonia, A. Singhal and J. Balwani, "Analysing Security and Software Requirements using Multi-Layered Iterative Model," *IJCSIT International Journal of Computer Science and Information Technology*, vol. 5, no. 2, pp. 1283-1287, 2014.

[31] K. Beck, M. Beedle, V. Bennekum and A. Cockburn, "Manifesto for Agile Software Development," *http://AgileManifesto.org*, 2001.

[32] A. Avizienis, J.-C. Laprice, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transaction on Dependable and Secure Computing*, vol. 1, pp. 11-33, 2004.

[33] C. Pfleeger and S. Lawrence, Security in Computing, New Jersey, USA: PRENTICE HALL, 2003.

[34] Y. Michael and T. Issa, "Properties for Security Measures of Software Products.," 2007.