Towards Secure Agile Software Development Process

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 challenges of agile practices that relate to the core "embrace-changes" principle of agile. Specifically, we analyze agile practices to find the security challenges in customer and developers activities. The argument of this paper is that changes to software are an important factor for security challenges and identify challenges for the "embrace- changes" that yield new security insights in the context of agile practices. Our case study based result show that a number of developers- and customer activities result security flaws and vulnerabilities into the software**.

Keywords*-*Agile; Embrace-changes; Development-process; Security-challenges; Security-principles

I. INTRODUCTION

Agile is one of the most popular approaches for software-development and its foundation is from its manifesto published by a group of software practitioners in 2001 [1]. The focus of agile is on developers and customers with the objective to produce working software quickly and accept changes throughout its lifecycle [2], [3], [4]. Therefore, agile is a widely used approach by software industries and is suitable for rapidly changing environments [5], [6], [7], [3], [8]. However, agile methodologies such as extreme programming (XP) and scrum do not support security because the focus of agile methodologies is on functionally working software and iterative delivery [6], [9]. At the same time, security is a critical part of software systems.

Our aim is to analyze agile practices in order to identify security challenges based on the security principles defined in [10]. Experience of practitioners, shows that security principles can guide the design and implementation of software without security flaws. We conduct case study based research [11] on the development process of applications that follows agile practices to analyze the relationship between security principles [10] and security challenges of agile practices. Our special attention is to identify security challenges in the activities of developers and customer of agile software development. For agile software development to remain agile we not introduce new practices, instead we evaluate agile practices for identifying security challenges. Our aim is refined to the main research question of our paper, namely: How to identify security challenges during changes to software? To establish a separation of concerns, the main research question is divided into the following sub questions: What are security challenges of response-to-changes based on security principles? What are the most frequent security challenges in agile software development? Which agile practices have more security challenges?

The remainder of the paper is structured as follows. Section II, provides related works and additional information relevant for agile practices, software security and security principles. Section III describes our case study design, data collection and analysis procedures. In Section IV we present the result of our research and evaluation of its validity. Finally, Section V concludes this paper by summarizing the research work, giving the contributions achieved and showing directions for future work.

II. RELATED WORK AND BACKGROUND

Literature detects deficiencies in agile methodologies due to the unavailability of security elements in their various phases and practices [12], [13], [6], [9]. Research results exits in the domain, but these research works lack a suitable approach to identify security challenges in agile practices, during changes-to-software. A group of researchers studies agile to explain- and analyze its behavior for security, in order to examine its practices for secure software development [12], [14], [13], [5], [15], [9]. Researchers also study security for the integration into a specific practice of agile methodology [16], [17], [18]. Other groups of researchers emphasize security to be included at each phase of a software development lifecycle [19] [8]. Furthermore, other researchers study agile security by introducing frameworks for the elicitation and analysis of security requirements [20], [21]. Agile practices, described in Section II-A, software security, described in Section II-B and security principles, described in Section II-C, are important elements in our research.

A. Agile Practices

The agile methodology is suitable for a rapidly changing environment and accepts changes to software at any phase of development. Changes to software are an important factor for security challenges and identifying them improves software security. Therefore, we study the relationship between changes-to-software, agile-practices, and security-principles.

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. Figure 1 shows agile practices.

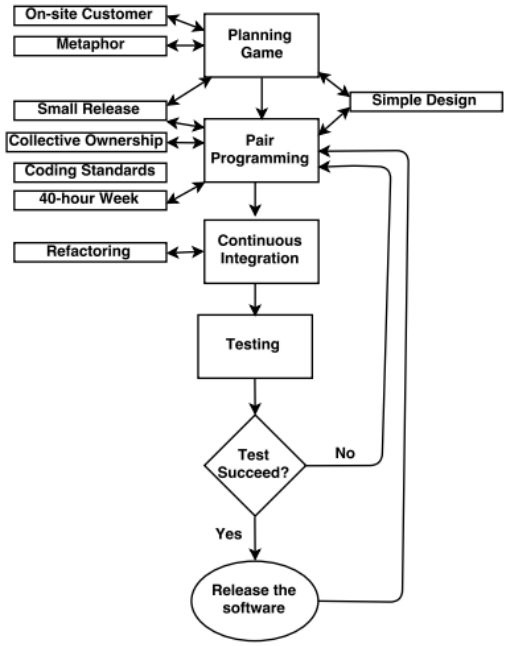

Figure 1: Agile Practices

The development process begins by requirements gathering and the main practice in this phase is the planning-game. 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 as well as negotiating user stories to be included in each scheduled release. Small-releases and simple design practices means it is up to the customer of the software to make important decisions. Small releases and short iterations help a project to continue with a sustainable progress rate. The planning game strategy is defined and prioritized according to customer needs and choices.

A development team implements 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. The developers determine the tasks necessary to implement the stories for the current iteration and estimate the amount of effort for performing each task as well as implementing all the stories in the current iteration. Consequently, an implemented feature is integrated to the software and continuous integration is the main agile practice in this phase. By 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. Tests check for the presence of all the features requested by the customers and assure the stability of the software in the presence of frequent changes. If the test succeeds, the updated software is put into use and the customers can decide about what new most important features should be added to the software.

B. Software Security

Security is a quality aspect of a system property that reflects the ability of the system to protect itself from accidental or deliberate attacks. Security is a composite of the attributes confidentiality, integrity, availability and accountability [22], [23]. 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 performed an operation.

C. Security Principles

Security principles are defined by [10] and can guide the design and implementation without security flaws. If some part of software development violates a principle, the violation is a symptom of potential flaw and the development process should be carefully revised to be sure that the flaw is accounted for. The following is the list of security principles:

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

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

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.

Least Common Mechanism: Minimize the amount of mechanism common to more than one user.

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

III. CASE STUDY

We choose a case-study based research method [11] to evaluate security challenges in developers- and customer activities of agile practices based on the security principles as listed in Section II-C. Case study in software engineering is an empirical inquiry that draws on the source of evidence to investigate a software phenomenon within its real-life context. A case study provides a deeper understanding of the phenomenon under study and is characterized by its flexibility and the conclusion is based on a clear chain of evidence. A case

study consists of these main phases: case-study design, data collection procedure, data analysis procedure and reporting [11].

A. Case Study Design

The software development process using agile practices depicted in Figure 1 is selected as a case for our study with holistic design [11]. We select four different software development teams in Kabul city and they are using agile practices for software development. The subject for our study is security challenges in agile practices. We conduct explanatory case study that has deductive features. Deductive case study starts with existing theories, sets out a set of hypotheses for the research, collects the evidence and finally compares to confirm or reject the theories. For guiding data collection and analysis, the following hypotheses are inferred from security principles, listed in Section II-C:

(i) Continuous changes-to-software make challenges for the process of separation of privilege.

(ii) Continuous changes-to-software increase the privileges for customer and developers.

(iii) Continuous changes-to-software affect negatively the developer attention.

(iv) Continuous changes-to-software increase the complexity of software.

(v) Continuous changes-to-software make it difficult to control the system-wide view of the software.

From eight security principles we derive five hypotheses. The first hypothesis is derived from the security principle "Separation of Privileges". The second hypothesis is derived from the security principle "Lest Privileges". The third hypothesis is derived from the security principle "Fail-safe Defaults". The principles of "Economy of Mechanism", "Psychological Acceptability" and "Open Design" are about the simplicity of software therefore, they are derived into the fourth hypothesis. The principles of "Least Common Mechanism" and "Complete Mediation" are both about system-wide view and control of software and both are derived into the last hypothesis. These hypotheses, derived from the security principles, guide the preparation of interview questions for gathering data about agile security. The result of analysis, either confirm or reject the hypotheses, which leads to either confirmed or rejected theories about agile security [11].

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 [10] 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 interview questions we checked with supervisor and other colleagues who have experience in case-study research. 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 Sources

The data collection method in our case study is interviews with software developers. We carried out interviews with 13 software developers in four different teams. All four teams use agile development methodology and each member of the team has at least experience of three software development projects. The interviewees were selected in such a way to cover the overall software development process. From each role at least two persons are included in the interviews.

The interview questions are derived from the hypotheses, listed in Section III-A, which are in an order defined from security principles, listed in Section II-C. The same questions are asked for the three main phases of agile practices, planning-game, pair-programming and continuous-integration that Figure 1 depicts. The mentioned three practices are collaborative and the activities of developers and customer in these practices are interdependent. The interview questions have an unstructured format that can provide additional insight beyond the interview questions. Each interview session lasts roughly 45 minutes to one hour. The interviews are audio recorded into WMA files for subsequent post-interview activities and analysis.

C. Analysis procedure

The main goal of analysis is to understand whether theories about the security challenges in agile practices are valid by testing the hypotheses, listed in Section III-A. For analysis, first the transcribed interviews are coded. The latter are meaningful labels organized by themes, or categories. These labels are assigned to phrases or sentences from the interviews. For coding and analysis, we use Nvivo[1] that is a qualitative data analysis software tool. To summarize coding, themes are introduced to group the codes. Each theme relates to a corresponding hypotheses from Section III-A. 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
Privileges & Responsibilities	Verify the identity of developers and customer based on their privileges and responsibilities.
Limitation of Privileges	Only necessary privileges, minimize interaction, & small components
Attention & Caution	Lack of access as default, deny access during mistake, & attention and caution
Software Simplicity	Make the design of software simple, small and easy
System-wide View & Control	Minimize common mechanism, check every access and deny access during a mistake

[1] http://www.qsrinternational.com/

The above themes are derived from security principles. During software development, if the activities of developers and customers are not in compliance with a process of inspection such as for design principles, then it can become a source for security flaws and vulnerabilities. We use a simple formula to evaluate what codes have more value for analysis. The formula is:

Code-value = (Sources * Phases) * Type

Sources denote how many different interviewees mention the code, phases denote the availability of code in the three main phases in Figure 1, and types show if a code is absolutely or conditionally mentioned by the interviewees. Each component in the formula has a numeric value and a higher value increases code validity. The possible values for phases are 1, 1.5, 2, 2.5 and 3. If the code is mentioned in one phase then the phase value is 1. If the code is mentioned in two phases and the type field for both phases is Abs (absolute) then the phase value is 2. If the second phase type value is Cond (conditional) then the phase value is 1.5. If the code is mentioned in three phases and the type field for phases is Abs then the phase value is 3. If the type value for the second or third phase is Cond then the phase value is 2.5. However, if the type value for the second and third code is Cond then the phase value is 2. The type value is 1 if the type field for the code is Abs and it is 0.5 if the type field is Cond.

Codes are sorted based on their value and then we review every theme separately and draw conclusions. All themes and codes are presented in Table II. The theme / code column contains themes and corresponding codes. We abbreviate the Phases and Type values. For phase value, c denotes planning-game, p denotes pair-programming and i denotes continuous-integration practice of agile. The type column values are respectively Abs for absolute and Cond for conditional. The value column gives the formula result.

Table II: Table of Coding Results

Theme / Code	Phases	Type	Value
1.Privileges & Responsibilities			
Customer puts responsibility on developers	c+p+i	Abs	32.5
Privilege & responsibilities of developers are not clearly documented	p+c+i	Abs	30
Different pairs & frequent changes to software	i+p	Abs	20
Different pairs cause unexpected errors	p+i	Abs	14
Customer gives unclear & unstable requirements	c	Abs	12
If tasks are not clearly specified	i+c+p	Cond	7.5
Customer is not well familiar with technology	c	Abs	7
If software is large with frequent changes	i	Cond	5.5
If developers don't make comments and do not work honestly	p+i	Cond	5.25
If developers have different ideas & knowledge level	p	Cond	5
If standards are not followed	i+p	Cond	2.5
Developers work based on customer interest	i	Abs	1
2. Limitation of Privileges			
Different pairs & frequent changes make it difficult to limit the privileges	i+p	Abs	20
Different pairs cause unexpected errors	p+i	Abs	14
Developers work for customer interest & customer changes ideas frequently	c	Abs	12
If tasks / responsibilities are not specified	p+i	Cond	9
Frequent changes of pairs & lack of documentation	p+c	Abs	7.5
If proper documentation is not provided	c+p	Cond	4
Frequent changes & repetition of work	c	Abs	3
If pairs do not have good relationship between them	p	Cond	2
If standards are not followed	p	Cond	1
3. Developer Attention			
Inconsistent feedback, idea & priority by customer cause repetition of work	c+i	Abs	22
Different pairs & frequent changes to software	i+p	Abs	20
Different pairs & frequent changes to software	p+i	Abs	14
Customer wants software quickly	c+p	Abs	13.5
Customer changes requirements & developer focus is on changes	c	Abs	9
Repetition of work & pressure on developers	i+c	Abs	8
If developers have less time	p+c	Cond	7
Customer cannot explain data protection requirements	c	Abs	5
If developer is not sure about his idea and depend on other ideas	p	Cond	4.5
If developer only express his idea and others write code	p	Cond	3
If working in large software for long time with frequent changes	i	Cond	3
If software has error and that needs to be fixed	i	Cond	2
4. Software Simplicity			
Customer gives unclear & unstable requirements	c	Abs	12
Different pairs & frequent changes to software	i	Abs	10
No problem	p+c	Abs	8
If frequent & inconsistence changes	c	Cond	6
No problem	c+p	Abs	4
If pair don't follow the standards	p+i	Cond	3.75
If developers have different idea	p	Cond	3
If the scope of software is larger	i	Cond	3
If frequent & inconsistence changes	i	Cond	2.5
Developers work for customer interest	c	Abs	2

If each pair works on separate part	p	Cond	2
Lack of proper documentation	i+p	Abs	1.5
If not revise the structure of software	i+p	Cond	0.75
If order of integration is not logical	i	Cond	0.5
5. System-wide View and Control			
Different pairs & unstable status of software	i+p	Abs	18
Frequent changes & separate integration	i+c	Abs	16
Customer gives unclear & unstable requirements	c	Abs	12
Customer determines the scope & priority	c+i	Abs	12
If pairs don't know the work of others	p+i	Cond	7
Setting priority by customers make it difficult to control the sequential logics of software	i+c	Abs	6
Frequent changes also change the structure	c+i	Abs	4
We lose the overall view for small components & details	p	Abs	3
If the scope of software is larger	i	Cond	3
If standards are not followed	p	Cond	2.5
If too many changes in short time	p	Cond	1.5
Customers has low technical knowledge	c	Abs	1
If pairs don't take responsibility for system-wide view	p	Cond	1

IV. RESULTS

Software security needs a variety of security mechanisms from gathering the requirements, implementation, testing and using environment. We are focusing on one aspect of software security that is the developers- and customer activities. From the earlier stipulated goals in Section 1, we derive our hypotheses from security principles and then for each hypothesis, we determined a corresponding theme. Through the coding session, the collected data from interviews is organized into corresponding respective themes. Following is the result of comparing the coded data in Table II, together with the corresponding security principles in order to answer our research questions.

Our first theme belongs to the security principle of "Separation of Privileges". Based on this principle, the secure software development process must verify the identity of developers and customers based on their privileges and responsibilities. However, the first high ranked code in the corresponding theme indicates the unclear privileges and responsibilities between customers and developers. This can compromise the "accountability attribute" of security, described in Section II-B. The coded data in the theme "Privileges & Responsibilities", gathered from interviewees also shows that the privileges & responsibilities between developers are not clearly documented. Looking at Table II, the problem of unclear privileges and responsibilities between customers and developers is mentioned in the three main phases and the highest rate of the problem is in the planning-game practice. The second highest ranked problem is the frequent changes to software by different pairs that make it difficult to verify the identity of the developers. For security purposes, controlling who performs changes to software is very essential. This problem relates to the pair-programming and continuous-integration phases of agile. The high rate for this problem belongs to the pair-programming phase.

Next, our second Theme we derive from the security principle of "Least Privileges". Based on this principle, the objective for developing secure software is to only provide the necessary privileges for developers and customers. Applying this principle on developers- and customer activities then if a question arises related to misuse of a privilege, the number of entities that need to be inspected is minimized [10]. Violations of this principle makes access control difficult and can compromise all the related attributes of software security. Looking to the theme "Limitation of Privileges" in Table II, the first high ranked code in the corresponding theme indicates that different pairs with different knowledge levels in agile teams exist, while making frequent changes to the software is the main challenge for the limitation of their privileges. This challenge relates to the pair-programming and continuous-integration phases and the highest rate of the problem is in the continuous-integration phase. The second high ranked problem in this theme is the dependency of developers on the unstable ideas of customers that extend the privileges of customers. This problem belongs to the planning-game phase of agile. Frequent changes of pairs and members of the pairs are also problematic for the limitation of the developer's privileges in the pair-programming and continuous-integration phases.

Our third theme of "Attention and Caution", 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. Certifying that the software is actually implemented as intended, particularly for security consideration, needs precise attention and caution from the developers. Based on our collected data in Table II, developers and customer activities can negatively affect the developer attention that consequently may introduce flaws and vulnerabilities into the software. As we can see in Table II, the most mentioned code in this theme is the inconsistent feedbacks, ideas and prioritization of tasks by customers that cause repetition of work for the developers. This repetition of work negatively affects the developers' attention because the unstable ideas of customers require more changes and repetition of work that cause pressure on developers to focus on changes. This is identical with the aim of the agile software development methodology to produce functionally working software with iterative delivery [13], [6]. However, it is contradictory with the developer attention to security issues of the software. This problem is related to the planning-game and continuous-integration phases of agile. The second high rate code in theme "Developer Attention" is different pairs and frequent changes to software under time pressure that also

indicate the problem for developers' attention. This problem pertains to the pair-programming and continuous-integration phases of agile and the high rate of the problem belongs to the continuous-integration phase of agile.

The fourth theme we derive from "Economy of Mechanism", "Open Design" and "Psychological Acceptability" security principles. Based on these principles and in order to apply the protection mechanism effectively, the design must be simple and small since techniques such as line-by-line inspection for finding security flaws in the code of software are necessary. For such techniques to be successful, a small and simple design is essential [10]. Although the simplicity of software is taken into account by the agile practices of "Simple Design" and "Small Release", the nature of activities of developers and customer are interdependent and there is no guarantee that the developers and customer will adhere to these practices. The interviewees confirm that the unstable requirements from customers need frequent and inconsistent changes to software that increases the complexity of software. This challenge belongs to the planning-game practice of agile. The second highest ranked code that shows the increase of complexity in software of this theme is pertains to the working of different pairs on the software that comprises frequent changes to software by these pairs. This problem relates to the continuous integration practice of agile. This code is also backed by another code in this theme and states that developers work on behalf of the customer interest and frequent changes in customer ideas cause illogical sequences of integration to the software that increases the complexity of the latter. This code belongs to the planning-game phase of agile.

Finally, the last Theme belongs to "Complete Mediation" and "Least Common Mechanism" principles of security. Based on these principles, security attributes have a system-wide nature and the protection and authorization mechanisms for developing secure software, requires the prevention of all unauthorized activities during software development. In other words, the effect of every change must be checked for the whole software. These requirements have negative form which make hard to prove that this negative requirements have been achieved since negative requirements require the anticipation of all possible flaws and vulnerabilities [10]. Consequently, developers must demonstrate that every possible threat has been anticipated during the development process. Thus, an expansive view of the problem is most appropriate to ensure that no gaps appear in the whole software during the development process. However, the highest codes in the corresponding theme, for these principles shows that different pairs, frequent changes and separate integration make the status of software leads to unstable software resulting in difficulties for developers to keep the system-wide view and control of software. This challenge relates to the planning-game, pair-programming and continuous-integration phases of agile. The second highest ranked code in this theme is that customers state unclear and unstable requirements. This is also backed by the third highest ranked code where interviewees indicate in agile, the scope of software and related priority of tasks are determine by customers that poses a challenge for developers to control the sequential logic of software. This challenge belongs to the planning-game and continuous-integration phases of agile.

V. Conclusion and Future Work

In this paper, we conduct a case study to identify and explain security challenges of agile software development by evaluating developers and customer activities based on security principles. 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 challenges and agile practices based on security principles [10]. The result of our study shows that, a number of developers- and customer activities, introduce security flaws and vulnerabilities into the software. For developing secure software using agile, our study show that, in order to keep the agile way of software development we need a new type of agile tool support. Table III shows security challenges, their frequent occurrence and relationship to agile practices. In this table, X denotes the existence of the challenge and X-H denotes that the challenge is higher in the corresponding phase then the other two phases.

Table III: Security challenges, their frequent occurrence and relationship to agile practices

Challenges	Categories	Agile Phases		
		Planning Game	Pair Programming	Continuous Integration
Unclear privileges & responsibility between customer & developers	Separation of Privileges	X-H	X	X
Unclear privileges & responsibility between developers	Separation of Privileges		X-H	X
Frequent changes & different pairs	Separation of Privileges		X-H	X
Frequent changes & different pairs	Limitation of privileges		X	X-H
Dependency of developers on customer ideas	Limitation of privileges	X		
Frequent changes of pairs & lack of documentation	Limitation of privileges		X-H	X
Inconsistence feedback & idea of customer	Developer attention	X-H		X
Tasks priority by customer	Developer attention	X-H		X
Different pairs & frequent changes	Developer attention		X	X-H
Unstable requirement from customer	Software simplicity	X		
Different pairs & frequent changes	Software simplicity			X

Illogical sequence of integration	Software simplicity	X		
Different pair & frequent changes	System-wide view	X	X	X-H
Unclear & unstable requirements	System-wide view	X	X	X-H
Scope & task priority by customer	System-wide view	X-H		X

Table III shows the three high ranked challenges, based on their code value in Table II, for each theme. As we can see in Table III, the challenge of "Different pairs and frequent changes" relates to every theme. The second highest ranked challenge that belongs to four different themes is the "Unclear and inconsistent requirement and frequent changes in customer idea". Software scope and tasks priorities by customer cause illogical sequence of integration come third grade most frequent challenge.

Looking to the relationship of these challenges to agile software development phases and practices in Table III, we can find that challenges such as "Unclear privileges and responsibilities between and developers" and "Different pairs and frequent changes" are the challenges that relate to all three main phases of agile. Most other challenges relates to the "Pair-programming" and "Continuous-integration" phases of agile. Later on, the third high category challenges relates to the "Planning-game" and "Continuous-integration" phases of agile. Looking to Table III from other perspective we can find that the first, second and third high ranked phases for security challenges are "Continuous-integrations", "Planning-game" and "Pair-programming" phases of agile respectively.

To develop secure software, the unclear and inconsistent ideas and requirements of customers as well as tasks priority by them need to be compliance with the security principles. For developing secure software the amenability between security principles and the interdependent work nature of developers in different pairs, frequent changes and separate integrations is also required. In current nature they are not obedience with security principles and can make problem for the security requirements such as authentication and verifying the identity, access limitation, developers' attention, software simplicity and system-wide view and control of software. Based on the security principles, the aforementioned security requirements are essential concerns for developing secure software system.

As a limitation of this research, the interviewed developers have little knowledge about software security and we are not able to design our interview questions to directly address software security. Instead, we derive the interview questions based on the security principles [10] 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 developers.

REFERENCES

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

[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] 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.

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

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

[8] Microsoft, "Microsoft Security Development Lifecycle for Agile Development," *http://www.microsoft.com/sdl.* , 2009.

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

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

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

[12] 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.

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

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

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

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

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

[18] 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.

[19] D. Owens, "Integrating Software Security into the Software Development Lifecycle System Securities".

[20] E. Mathisen and T. Fallmyr, "Using Business Process Modeling to Reduce the Effects of Requirements Changes in Software Projects," in *IEEE*, 2009.

[21] C. Haley, L. Robin, M. Jonath and N. Bashar, "Security Requirements Engineering: A Framework for Representation and Analysis," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 34, pp. 133-153, 2008.

[22] 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.

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