Towards a Secure SCRUM Process for Agile Web Application Development

Patrik Maier
Institute of Applied Information Processing and Communications
University of Technology
Graz, Austria
patrik.maier@student.tugraz.at

Zhendong Ma
Center for Digital Safety & Security
Austrian Institute of Technology
Vienna, Austria
zhendong.ma@ait.ac.at

Roderick Bloem
Institute of Applied Information Processing and Communications
University of Technology
Graz, Austria
roderick.bloem@iaik.tugraz.at

ABSTRACT

Agile development such as Scrum and Extreme Programming deliver software in short iterations for quick response to rapid business requirement and market changes. However, established secure software development methodologies are mostly based on linear models such as waterfall and V-model, making them unsuitable for direct application in an agile environment. This paper presents a proposal for integrating security activities into Scrum process for developing secure Web applications. We identify gaps in existing approaches to secure agile development and analyze established security engineering activities. We then adapt these activities and orchestrate them into Scrum development process to achieve both security and agility. Our proposal is evaluated by a Scrum team developing commercial JAVA EE applications in an opinion survey.

KEYWORDS

secure Scrum, agile development, Secure Development Lifecycle (SDL), Web application security

ACM Reference format:
Patrik Maier, Zhendong Ma, and Roderick Bloem. 2017. Towards a Secure SCRUM Process for
Agile Web Application Development. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 8 pages.
DOI: 10.1145/3098954.3103171

1 INTRODUCTION

Agile software development advocates short iteration cycles, early delivery, and incremental software development to enable software projects to quickly react to changes in business requirements [4]. It is fundamentally different from traditional linear development models such as waterfall and the V-model. Scrum is a framework of practices for agile development, in which software is incrementally implemented, tested, reviewed, and shipped in Sprints, usually lasted for 30 consecutive calendar days or less. Due to dynamic customer needs and the pressure on time-to-market, agile development is used extensively in Web application development.

However, Web application has a large attack surface and is a popular target of remote attacks on the Internet. Statistically they exhibit a track record of high number of vulnerabilities and incidents in the past [14]. Agile development makes Web application security even more challenging. Many established software assurance practices [5, 7, 9] are based on linear development models. The identification of threats and security requirements and the creation of secure software architectures all require a considerable amount of time, which contradicts with the principle of agile development. In an agile development, software is implemented in increments. It accepts the fact that requirements will often change and hence avoids time-consuming planning phase. It also minimizes the effort on documentation and modeling. The "agile" practices are not totally compatible with traditional software security practices emphasizing on careful planning, thorough analysis, and iterative design and development. Hence the main challenge of achieving Web security in Scrum process is to balance the efficiency and dynamics of agile development with security with appropriate cost-effectiveness.

Several approaches have been proposed in the past to address secure Web application development in an agile environment, especially for Scrum [18]. A majority of existing work focus on integrating security activities into the development process or extending Scrum activities and artifacts to cover security concerns. Although Web security in Scrum is a multifaceted problem that requires solutions from many aspects, we argue that a low hanging fruit is a secure development process that can be readily integrated into existing workflow of a development team in software production. In this paper, we propose a secure Scrum process for Web application development. Specifically,

- we propose a secure Scrum process that maps and integrates security engineering practices from the ISO standard Systems Security Engineering – Capability Maturity Model (SSE-CMM), originally designed for linear development;
- we propose an agile risk analysis method that balances the agility and effectiveness of security analysis in agile development;
- and we evaluate our approach with developers who use Scrum on a daily basis for commercial software.

Our focus is on Web application, mainly because agile method is widely used in Web development. Another reason is that we have the possibility to evaluate our approach with a Scrum team developing commercial Web applications. We use OWASP top 10 security risks [15] as a benchmark for Web application security. The goal is to reduce vulnerabilities and security bugs in Web applications developed in Scrum process. In the following, Sec.2 gives

ARES '17, Reggio Calabria, Italy
© 2017 ACM. 978-1-4503-5257-4/17/08…$15.00
DOI: 10.1145/3098954.3103171

background information about Scrum followed by a discussion of related work in Sec.3. Sec.4 presents our secure Scrum process and Sec.5 evaluates our approach, followed by the conclusion in Sec.6.

2 OVERVIEW OF SCRUM

Scrum [18] is an agile software development process. A Scrum team consists of a product owner, a Scrum master, and a development team. The product owner defines the objectives and requirements of the project and the release plan. A product backlog is used to include all defined functional and nonfunctional requirements. The Scrum master helps the development team to follow the Scrum practices and provides coordination between the product owner and the development team. A development team ideally consists of three to nine people, with the role of implementing the software. All members of a team are cross-functional and equal, i.e. no strict roles such as testers or designers exist. Everyone within the team can conduct software design, implementation, and testing.

Software is incrementally implemented in Sprint, initiated by a Sprint planning. It is divided into two parts. In the first part, the product owner and the development team define the most important requirements that will be turned into functionality in a single Sprint. In the second part, the development team identifies the tasks necessary to transform the requirements into functionalities and plans how to actually achieve it. The result is documented in a Sprint backlog. Every day in a Sprint, the development team holds a daily Scrum meeting that takes no more than 15 minutes to exchange and discuss progresses and problems. New items in the product backlog can be created in product backlog refinement session, held between the product owner and the development team. The session can also be used to refine the items already in the product backlog in more details. The time spent on product backlog refinement should not exceed 10% of the capacity of the development team. A Sprint review meeting is held at the end of the Sprint. The increment implementation is assessed according to the Sprint goals defined in the backlog. All items in the sprint backlog should be completed at the end of a Sprint. Completeness is defined by the "Definition of Done", which is a list of requirements to be met for the software increment. Items that do not obtain the status of "Definition of Done" are returned to the product backlog and re-prioritized.

User stories [2] are commonly used in Scrum practice. A user story consists of a one sentence summary of the business value and several acceptance criteria, which are written on a physical form, such as a sticky note. A development team defines individual software increments in user stories together with its customer or the product owner. For defining a user story, the Independent, Negotiable, Valuable, Estimable, Small, and Testable (INVEST) [1] principle is used. A user story must fulfill the "Definition of Ready" (i.e. a list of criteria to be met before a user story becomes immediately actionable) before it can be selected for a Sprint planning for later implementation. A Scrum release plan describes the plan for multiple Sprints. It is a list of requirements or user stories which should be shipped in a release.

3 RELATED WORK

Existing work mainly focuses on adding security activities and techniques into agile development process or extending agile activities or artifacts to cover security concerns. For example, in extreme programming (XP), an XP team must follow development practices such as simple design, testing, re-factoring, metaphor, collective ownership, coding standard and pair programming. Wäyrynen *et al.* [22] analyzed XP's suitability for developing secure software and used the SSE-CMM as a checklist for evaluating which practices are already taken into account. Their analysis shows that XP lacks support for the assessment of security risks and specification of security needs because security requirements are non-functional, which cannot be easily broken down into estimable tasks. Therefore, security experts need to be included in XP for risk assessment and for pair programming with the development team. Microsoft extends its Secure Development Lifecycle (SDL) to cover agile development [13]. Although it has the potential to become another *de facto* industry standard, at the currently form it is only a collection of best practices recommendations.

User stories are a type of artifacts used in Scrum to define software requirements and increments. Several approaches have proposed to use user stories for security in Scrum. Asthana *et al.* [3] proposed to use security-related user stories to identify security threats and define security requirements in security-related user story templates. The templates can be used for security requirements which are then divided into small manageable tasks. Tuuli *et al.* [19] also proposed that a product owner should write specific security-related user stories with the help of security-related user story templates. These stories are then implemented by the development team similar to ordinary functional ones. From a development process point of view, such an approach has the advantage of easily integrating security requirements into current activities that the developers are familiar with. Pohl and Hof [17] extended this approach in which security concerns of a functional user story is formulated as a security-related user story, misuse story, or abuse story. The concerns are linked to corresponding functional user stories in the product backlog. Hence, security concerns of a functional user story become product backlog items and are linked to functional user stories. "Definition of Done" is required to include the verification of security concerns. A new backlog item covering the verification part is added to the product backlog if verification fails.

Several techniques to replace time-consuming security analysis have been proposed. Vähä-Sipilä [21] proposed to use generic "security story" templates and threat modeling to identify abuse cases and threats. The requirements are then broken down into estimable tasks and added to the product backlog. Threat modeling and other security tasks are performed in Sprints and added as backlog tasks in order to allow a product owner to measure costs.

4 SECURE SCRUM PROCESS

4.1 Methodology

We aim at designing a secure Scrum process to achieve both agility and security in Web application development. Assuming that most security activities will introduce overhead in terms of time and additional human resource, our consideration hence focuses on what type of security activities are essential to security, what security practices brings the most benefit with the least overhead, and whether we can minimize the overhead introduced by these activities.

Inspired by [22], we use the processes described in SSE-CMM [7] as a baseline to select and integrate a subset of the security activities for Scrum. SSE-CMM is an ISO/IEC standard for secure software development. It introduces security engineering practices into the development lifecycle and describes how to evaluate the capability level of such practices. We regard it as a comprehensive collection of security engineering activities to date. SSE-CMM organizes best practice security activities into 11 process areas. In order to select the security activities with the most cost-benefit in the context of Scrum, we analyze the existing security activities given in SSE-CMM and cross-check them with additional well-established secure software development methodologies such as NIST 800-64 [9], Security Requirements Engineering Process (SREP) [12], Cigital Touchpoint [10], and Microsoft SDL [5] to identify common denominators. Based on the analysis, we propose a secure Scrum process that integrates security-related activities, tools, and quality gates. In addition, risk analysis is an essential security activity. To reduce the overhead introduced by traditional risk analysis method, we design an agile risk analysis method to be an integral part of the secure Scrum process. A detailed discussion of these activities is given in Sec. 4.3.

4.2 Analysis of agility of existing security engineering activities

The objective of the analysis is to identify specific activities, methods, tools, and techniques that can be integrated into a secure Scrum process. These security activities are conducted in the phases of security requirement specification, software design, secure software implementation, and verification.

We adopt the criteria of agility given in [8], which includes simplicity, free of modeling and documentation, tolerant to requirement changes, minimal speed of execution, people oriented, informality, iterative, and high flexibility. The original definition is abstract. We extend the criteria with more details. For example, "simplicity" for us means to ask "what kind of security expertise must the person have in order to complete it in an effective way?", "tolerant to requirement changes" means "is it necessary to wait for the completion of other activities in order to be able to complete an activity?" and "how much has to be redone on small or medium requirement changes?" We interpret "high flexibility" as "is it possible to skip some steps without being ineffective, due to, for example, time reasons? Can these steps be replaced by other ones which are more efficient? Should all steps be performed strictly as described to be effective?"

To quantitatively measure the agility of each of the activities, we define a metrics in which each aforementioned element of agility is further given a value in the scale from 0 to 5. For example, for "simplicity", 0 means the least simple and 5 means the most simple. We assume it will take on average 2 weeks to implement a user story in Scrum sprint. Hence 0 for "speed of execution" means it will last for the whole Sprint, and 5 means that it will only last 1-2 hours. Since there are 8 elements, the highest score an activity can get is $5 \times 8 = 40$. We arbitrarily define if an activity receives more than 21 points (above 50%) it is "agile compatible". We also regard an activity to "agile friendly" if it receives more than 26 points (above 67%). Note that the decision on the cut-off values is intended to gain relative measures of agility.

We review the security activities specified in NIST 800-64, SREP, Cigital Touchpoint, and Microsoft SDL and evaluate them against the 8 elements of agility. Our evaluation identifies 7 activities that have the scores qualified to be "agile friendly", i.e. security requirements analysis, role matrix, static analysis, dynamic analysis, code review, initiate security planning, and conduct testing. Moreover, we also identified another 7 activities that qualified to be "agile compatible", i.e. attack surface analysis, abuse cases, penetration testing, critical assets, repository improvement, categorize information system, and assess privacy impact. Interested readers are referred to the corresponding original documents for detailed background information.

4.3 A proposal for secure Scrum process

The secure Scrum process is a collection of security activities organized in a balanced way for both security and agility. For security, our rationale is to integrate the security activities identified as "agile". Our first step is to group these activities into requirement, design, implementation and verification phase similar to a waterfall model. Then we adapt these activities to be used in the secure Scrum process. A description of our adaptation of these security activities is given below.

4.3.1 Requirement.

In *Initiate Security Planning*, the software company discusses with the stakeholders of the project about security policies and high-level security requirements. Security visions and roles are defined. The participants should obtain a basic knowledge of security expectations. The meeting should be documented and a provisional schedule and milestones should be defined. The goal of*Agile Risk Analysis: External Dependencies* is to define the deployment environment and security assumptions. Security assumptions should be linked to corresponding security-related user stories. *Security Requirement Analysis* identifies security requirements and documents them in the security section of the user stories or epics (i.e. a work that takes longer than a week or a full Sprint to complete). Furthermore, possible attack scenarios can be added to the user stories. Note that it is assumed that not all security requirements and attack scenarios can be determined in this stage. It obtains a high-level understanding of security requirements. Security requirements are kept as acceptance criteria in functional user stories. *Agile Risk Analysis: Identify Trust Level* identifies all users who have access rights to the application. It also creates attacker profiles and establishes a high-level understanding of application access. *Agile Risk Analysis: Identify Assets* lists all assets categorized by trust levels. The information is stored with their impact levels. *Agile Risk Analysis: Determine Entry Points* identifies unnecessary user accesses to the assets described in the user story.

4.3.2 Design.

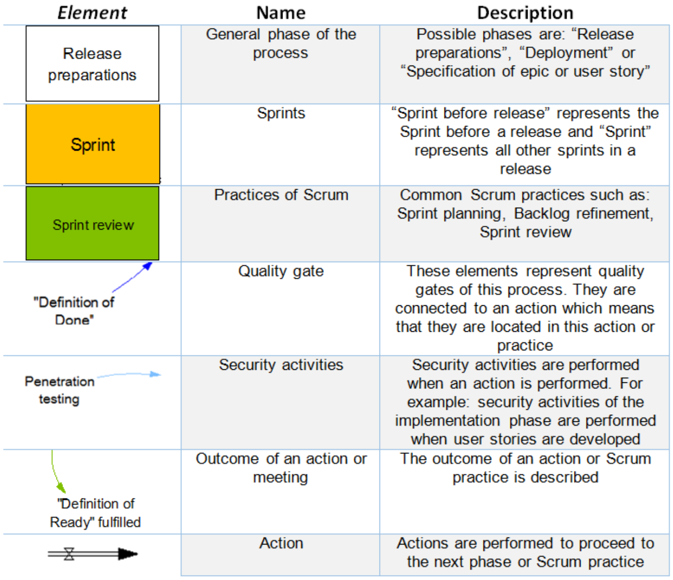
Agile Risk Analysis: Identify Threats identifies concrete threats in the context of a specific user story. High-level threats identified in previous steps are analyzed to determine whether they are specific enough for a story. For each identified threat, a security-related user story and abuse cases are defined and attached to the functional user story. In *Agile Risk Analysis: Ranking of Threats with DREAD*, the results are presented to the team in another product backlog refinement meeting. The product owner should discuss with the whole team about the impact of each threat. The Damage potential, Reproducibility, Exploitability, Affected users, Discoverability (DREAD) [11] method can be used for ranking threats. If the ranking cannot be completed for a user story, it will not obtain a "Definition of Ready". Hence, the user story has to be refined again in another product backlog refinement meeting. In *Agile Risk Analysis: Mitigation Strategies*, security controls are chosen to each threat as mitigation strategies in a product backlog refinement meeting and documented in the corresponding security-related user stories.

4.3.3 Implementation. In *Document Security Controls*, the developers tailor the high-level security controls to fit for a specific user story. They describe the reason for the design decision for a specific security control in the corresponding security-related user story. The documentation should be a short formal description. *Static Code Analysis* requires each developer to perform security vulnerability scans with static code analysis tools over their code. If issues are found in the code, a security bug is immediately reported to a bug tracking system, such as Jira. *Dynamic Code Analysis* uses approved dynamic code analysis tool for frequently testing of the software. We recommend *Pair Programming* for security-critical software components, i.e. one developer writes the code whereby the other one looks for coding failures or security flaws at the same time. It means that at lease on developer has sufficient security expertise to recognize security flaws. In every Sprint, *Dependency Checker* tools are used in order to check if libraries contain publicly-known security vulnerabilities. In case of insecure libraries, a security bug is reported to a bug tracking system. If libraries with known issues are used, it is documented why it is not possible to use another library or whether the known issues are relevant.

4.3.4 Verification. Verification should be risk-based. Agile risk analysis should be used to prioritize the definition of test cases and reviews according to the risks. In *Pair Penetration Testing*, a quality engineer with the developer who generated the code should test the program using a black box Web vulnerability scanner. If possible, only the new implemented functionality should be tested with this tool. If the result contains false positives, all false positives should be documented and explained why they are not security flaws. The result is used to speed up testing in penetration testing. In *Penetration Testing*, a penetration tester tests various attack scenarios either manually or automatically with all-in-one penetration testing tools. The penetration tester should use the list of false positives from the previous activity to avoid exploiting the same vulnerabilities. Issues detected are prioritized with DREAD and handled in the activity *Agile Risk Analysis: Mitigation Strategies*. In *Code Review*, a security tester performs a security code review with the developer for the user story. In this review they manually review the source code to identify security flaws.

Besides these activity groups, quality gates are used in the development process to describe milestones and to evaluate whether all obligatory criteria are fulfilled as the condition for proceeding to the next phase. Quality gates are placed between phases. As an existing mechanism in Scrum for quality control, it provides an opportunity for security-related reviews.

4.3.5 Secure Scrum.
Fig. 1 shows the components of a secure Scrum process and Fig. 2 gives an overview of our proposed secure Scrum process. The process is based on Scrum specification and the aforementioned security activities that we regard as agile enough to be integrated into the process. In the figure, standard Scrum practices (shown as green boxes) are extended with security activities (highlighted in red rectangles). We adopt the principle of Scrum release planning, which is a date- or feature-driven plan for multiple Sprints including a list of requirements or user stories to be shipped in a release [6]. We assume that a software company ships around four releases in a year. Each release consists of four Sprints and each one is performed in the time-span of two weeks. Note that in reality, there will be many variations in these numbers from company to company.



Element	Name	Description
Release preparations	General phase of the process	Possible phases are: "Release preparations", "Deployment" or "Specification of epic or user story"
Sprint	Sprints	"Sprint before release" represents the Sprint before a release and "Sprint" represents all other sprints in a release
Sprint review	Practices of Scrum	Common Scrum practices such as: Sprint planning, Backlog refinement, Sprint review
"Definition of Done"	Quality gate	These elements represent quality gates of this process. They are connected to an action which means that they are located in this action or practice
Penetration testing	Security activities	Security activities are performed when an action is performed. For example: security activities of the implementation phase are performed when user stories are developed
"Definition of Ready" fulfilled	Outcome of an action or meeting	The outcome of an action or Scrum practice is described
→	Action	Actions are performed to proceed to the next phase or Scrum practice

Figure 1: Components of secure Scrum process

We assume that a security specialist is needed to cooperate with the customer, product owner and the development team for security requirement analysis.

Besides the activities already described, addition considerations in the secure Scrum process are given below.

- Requirement. In current release n, the requirement phase of the next release $n + 1$ starts. In other words, in the last two Sprints of a release, the epics and user stories of the next release are planned. At the same time, high-level security requirements, trust levels, possible assets, and their entry points are identified. The results are included in a security section of the epics or user stories of release $n + 1$.
- Design. In the design phase, security-related technical user stories are identified. Product backlog refinement meetings

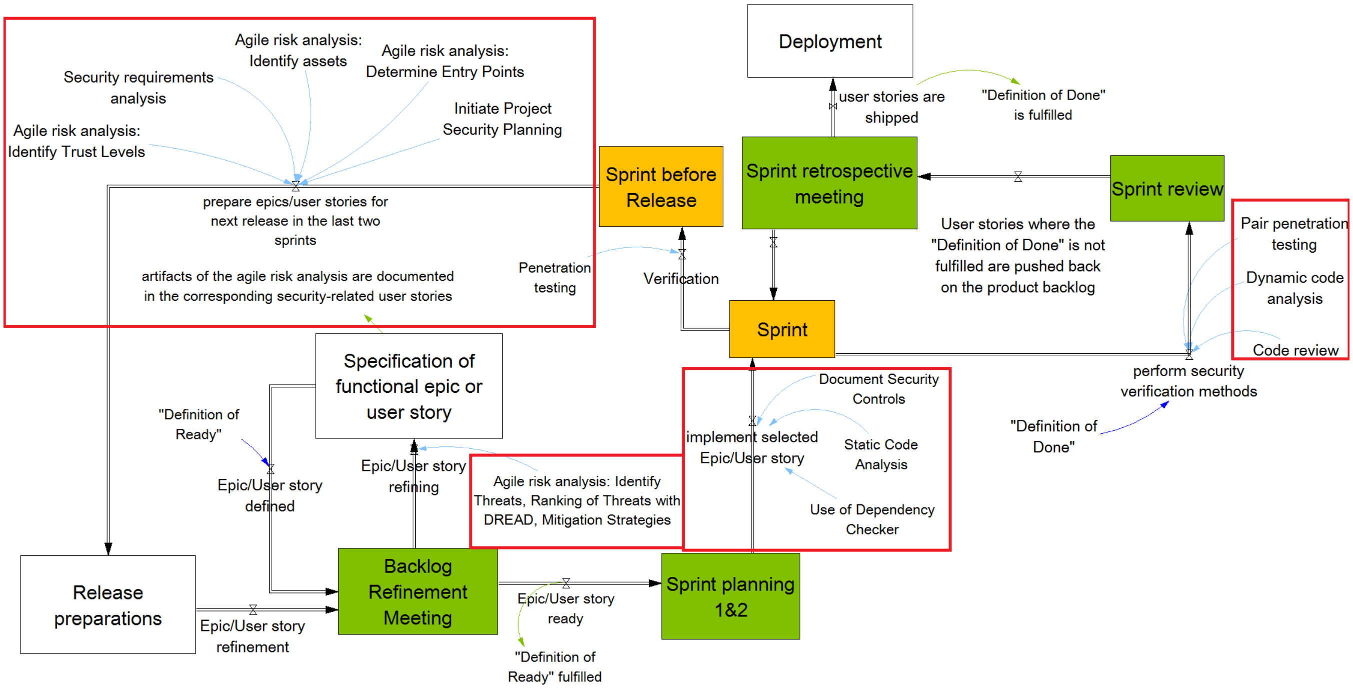

Agile risk analysis: Identify assets
Agile risk analysis: Determine Entry Points
Security requirements analysis
Initiate Project Security Planning
Agile risk analysis: Identify Trust Levels
prepare epics/user stories for next release in the last two sprints
artifacts of the agile risk analysis are documented in the corresponding security-related user stories
Penetration testing
Deployment
user stories are shipped
"Definition of Done" is fulfilled
Sprint before Release
Sprint retrospective meeting
Sprint review
Verification
User stories where the "Definition of Done" is not fulfilled are pushed back on the product backlog
Pair penetration testing
Dynamic code analysis
Code review
Sprint
perform security verification methods
"Definition of Done"
Specification of functional epic or user story
"Definition of Ready"
Epic/User story defined
Epic/User story refining
Agile risk analysis: Identify Threats, Ranking of Threats with DREAD, Mitigation Strategies
Document Security Controls
implement selected Epic/User story
Static Code Analysis
Use of Dependency Checker
Release preparations
Epic/User story refinement
Backlog Refinement Meeting
Epic/User story ready
"Definition of Ready" fulfilled
Sprint planning 1&2

Figure 2: Secure Scrum process

are held in Scrum to evaluate whether user stories or epics have been changed so they can be refined. Security requirements are considered in these meetings. New threats are identified when a functional user story is implemented. These threats are then formulated into security-related user stories. The quality gate “Definition of Ready” makes it mandatory. Thus only if all activities of the requirement and design phase for a user story have been completed, this user story is allowed to be added to the Sprint backlog and the implementation of it can be started.

- Implementation. It is similar to normal Scrum. After user stories have been added to a Scrum teamfis Sprint backlog, the stories are implemented. Security control are added to the Scrum process in the context of a user story and documented in the corresponding security-related user story. If no security controls have been developed, the reasons are documented in the corresponding security-related user story.
- Verification. Stories are tested with pair penetration testing, code reviews, whitebox security testing, and penetration testing. Security testing are based on the outcome of the agile risk analysis to prioritize the test cases and reviews with regard to their risks. After these activities have been successfully performed, the “Definition of Done” is fulfilled by the user story or epic. A Scrum retrospective meeting is held together with the Sprint review. After these meetings, user stories which obtained the “Definition of Done” are shipped to the customer just like in normal Scrum.

In order to integrate security requirements in a Scrum process, it is necessary to define and document them. We adopt the security-related user story templates from SAFECode [3]. However, a development team should refine the templates for its specific functional user stories. Each of the security-related user stories receives a security debt value and is added to the Product Backlog. A debt value is a numeric value used as a mechanism to force and track a development team to implement certain security requirements specified in the security-related user stories. If a team does not develop the security user stories in a Sprint, the security debt value of these stories is added to the teams security debt. A security debt threshold is arbitrary defined by the team. If the debt value exceeds a certain threshold, the team must develop these security-related user stories in a separate Sprint. Security-critical user stories can also be added as an acceptance criteria. References should be added to link security-related user story to corresponding functional user stories as suggested in [17].

4.4 Agile risk analysis

Most existing risk analysis and threat modeling methodologies were originally developed for linear models and lack a certain degree of “agile friendliness. Nevertheless, security risk analysis and threat modeling is essential for software security. Therefore, we adapt the OWASP Application Threat Modeling (OATM) [16] and propose a risk analysis methodology tailored to agile development. We use OATM as a reference and remove and modify some of the activities, and replace some steps with agile friendly activities. The result is based on the agile friendly activities whereas it still covers the basic process areas in SSE-CMM for secure development. Specifically, the PAs considered are Assess Impact, Assess Security Risk, Assess Threat, and Assess Vulnerability. It consists of the following activities: Abuse cases, Role Matrix, Critical Assets, Attack Surface Analysis/Reduction, Categorize Information System. A description of the activities are given below.

- External dependencies and security assumptions- Define security assumptions and attach a link in the user stories. Use software dependency checker in every Sprint to find vulnerabilities in libraries that are used in a project.
- Asset. Identify the assets in a software and the user with access to them. A comprehensive identification of asset is based on either interviews with asset owners and stakeholders or functional requirements. Importance of assets can be ranked based on existing impact assessment matrix [20]. A table or database can be used to store and track all assets depending on the size of the project.
- Trust levels. Identify trust level of each of the user in which a matrix is used to clear identify which user is allowed for what kind of access (create, read, write, or execute) to each asset. It should be linked to user stories too.
- Entry Points. Identify entry points of potential attacks by using attack surface analysis activity [5].
- Threat identification. Brainstorming instead of threat analysis based on data flow diagram (DFD) typically used in Threat modeling is used. Threats are identify using security-related user story templates from Safecode [3]. These templates were created to support the formalization of security requirements in an agile development process. The simplification should significantly improve the agility of this step. Specific attack scenarios are mapped to abuse cases with the help of the OWASP Top10 document. These abuse cases should be formulated either formally or with the help of diagrams, which can be developed fast and should be attached to the corresponding security-related user story. This means that a security-related user story can contain one or more abuse cases. The goal is to identify further threats and to support the writing of specific test cases.
- Threat ranking. Ranking of threats using Damage potential, Reproducibility, Exploitability, Affected users, Discoverability (DREAD).
- Mitigation strategy. Design and implement mitigation following a set of strategies such as do nothing, remove the feature, turn off the feature, warn the user, implement security controls, or transfer the risks [16].

5 EVALUATION

We evaluate the secure Scrum proposal in two aspects: 1) the agility and cost-effectiveness of our proposal in terms of additional time to normal Scrum process, 2) the level of security it provides.

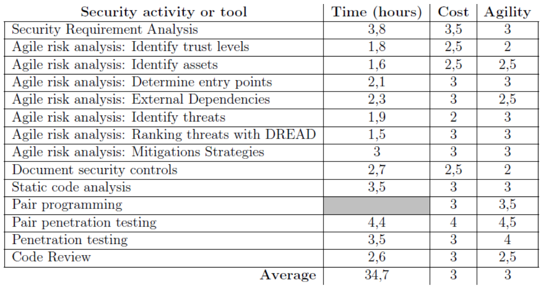
5.1 Evaluation of agility and cost-effectiveness

For the first aspect, we conducted a survey on a group of employees at a medium size software company using a questionnaire. The company develops commercial Java EE web applications for large corporate customers and follows the Scrum process. We explained the interviewee in an one hour presentation of the whole secure Scrum process and described each of its activities. After the presentation, we ask them to fill out a questionnaire on how agile and cost-effective they think each activity is. We also asked them how much time they would invest for completing each activity in the context of a user story. A group of ten people with mixed roles are involved in the survey, which includes four developers, two Scrum masters, one software architect, one project manager, one quality manager, and one product owner. The interviewees are selected based on the criterion that they are able to estimate the time, cost, and agility of a Scrum process from their specific perspectives.

Fig. 3 shows the median scores of all participants' answers. The time column displays how much time in hours the participants would spent in average to complete an activity. Note that the estimation on pair programming time is skipped, because we think that it is not possible for them to estimate the additional effort. The cost and agility columns illustrate the participants' median scores in regard to cost and agility to average out the subjective opinions. The maximum value in these columns is five, which represents that all participants have agreed that the activity is very agile. However, in contrast, if the value is one, all participants hold the belief that this activity is not agile-friendly. We use the following scale:

(1−1.49=very low, 1.5−2.49=low, 2.5−3.49=medium, 3.5−4.49=high, 4.5−5 = very high)

In summary, based on the evaluation of a group of developers, Scrum masters, software architects, project managers, quality managers, and product owners on the scale of "very low", "low", "medium", "high", and "very high", our secure Scrum process is regarded as "medium" agile and "medium" cost-effective.



Security activity or tool	Time (hours)	Cost	Agility
Security Requirement Analysis	3,8	3,5	3
Agile risk analysis: Identify trust levels	1,8	2,5	2
Agile risk analysis: Identify assets	1,6	2,5	2,5
Agile risk analysis: Determine entry points	2,1	3	3
Agile risk analysis: External Dependencies	2,3	3	2,5
Agile risk analysis: Identify threats	1,9	2	3
Agile risk analysis: Ranking threats with DREAD	1,5	3	3
Agile risk analysis: Mitigations Strategies	3	3	3
Document security controls	2,7	2,5	2
Static code analysis	3,5	3	3
Pair programming		3	3,5
Pair penetration testing	4,4	4	4,5
Penetration testing	3,5	3	4
Code Review	2,6	3	2,5
Average	34,7	3	3

Figure 3: Evaluation results on estimation of agility and cost-effectiveness

Furthermore, most participants agreed that security-related user stories should be added as product and Scrum backlog items.

5.2 Evaluation of security maturity level

The proposed secure Scrum process should be aligned with the SSE-CMM as a baseline for secure development process. In our approach, the secure Scrum process should take the Process Areas (PAs) of SSE-CMM into account. Fig. 4 illustrates how the Secure Scrum process handles the different PAs. We exclude PA 01 and 08, because they do not focus on the implementation and verification of secure software. To coordinate security we use the events of

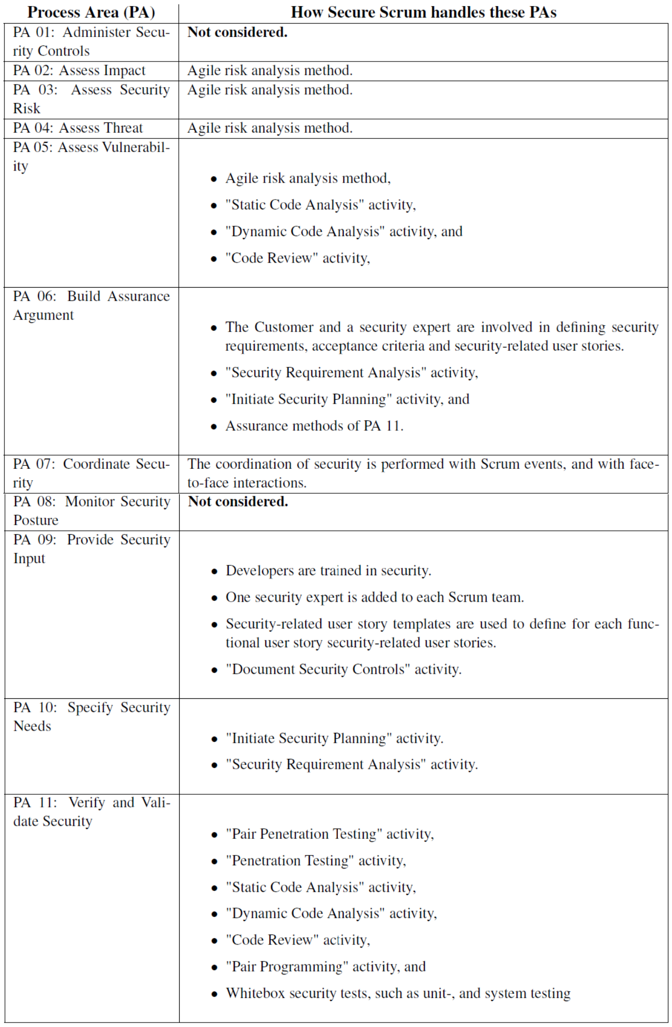
Scrum and the frequent face-to-face interactions (cf. Fig. 2). We use our agile risk analysis to achieve the goals of PA 02-05. In this methodology we identify threats and vulnerabilities of the implemented software, assess the impact on assets and calculate the actual security risk with DREAD. Static- and dynamic code analysis as well as code reviews detect further vulnerabilities. In order to assure that the software is really what the customer wants (part of PA 06 and PA 10) in regard of security we involve the customer in defining the security milestones in the “Initiate Security Planning activity and in the “Security Requirement Analysis activity. In this way, it is assured that the requirements reflect the customers wishes. In order to identify proper security control alternatives to satisfy the security requirements defined (PA 09), we conduct our agile risk analysis and use SAFECode’s security-related user story templates, which list several tasks to be considered if a functionality is implemented. In the “Document Security Controls activity we select specific security controls and implement them. Finally, in order to assure that the selected security controls actually fulfill the security requirements defined (PA 06 and PA 11) we perform Pair Penetration Testing, Penetration Testing, Static Code Analysis, Code Review, and whitebox security testing. Additionally, we add one Security Risk Specialist and a Security Tester to each Scrum team and involve them in several security engineering activities in order to spread security knowledge among all developers.

6 CONCLUSION

Agile development such as Scrum is very popular among Web application developers. However, many existing secure development methodologies rely on time-consuming security practices, which contradicts the agile and incremental nature of Scrum. In this paper, we proposed a secure Scrum process that integrates proven secure development practices into agile Scrum process. We analyzed well-known secure development approaches and adapt the security activities to fit into agile development. The proposed secure Scrum process has been evaluated by a group of software developers in an opinion survey. Our result shows that on the scale of “very low”, “low”, “medium”, “high”, and “very high”, our secure Scrum process is regarded as “medium” agile and “medium” cost-effective. We align our secure Scrum process to the Process Areas in Systems Security Engineering - Capability Maturity Model (SSE-CMM) to ensure security. Our secure Scrum process covers all desired process areas (PAs) of SSE-CMM. We also proposed an agile risk analysis method as a trade-off between agility and security for threat identification.

ACKNOWLEDGEMENT

The research leading to this paper has received funding from the H2020-ECSEL programme under the grant agreement no. 692474 (AMASS).



Process Area (PA)	How Secure Scrum handles these PAs
PA 01: Administer Security Controls	**Not considered.**
PA 02: Assess Impact	Agile risk analysis method.
PA 03: Assess Security Risk	Agile risk analysis method.
PA 04: Assess Threat	Agile risk analysis method.
PA 05: Assess Vulnerability	• Agile risk analysis method, • "Static Code Analysis" activity, • "Dynamic Code Analysis" activity, and • "Code Review" activity,
PA 06: Build Assurance Argument	• The Customer and a security expert are involved in defining security requirements, acceptance criteria and security-related user stories. • "Security Requirement Analysis" activity, • "Initiate Security Planning" activity, and • Assurance methods of PA 11.
PA 07: Coordinate Security	The coordination of security is performed with Scrum events, and with face-to-face interactions.
PA 08: Monitor Security Posture	**Not considered.**
PA 09: Provide Security Input	• Developers are trained in security. • One security expert is added to each Scrum team. • Security-related user story templates are used to define for each functional user story security-related user stories. • "Document Security Controls" activity.
PA 10: Specify Security Needs	• "Initiate Security Planning" activity. • "Security Requirement Analysis" activity.
PA 11: Verify and Validate Security	• "Pair Penetration Testing" activity, • "Penetration Testing" activity, • "Static Code Analysis" activity, • "Dynamic Code Analysis" activity, • "Code Review" activity, • "Pair Programming" activity, and • Whitebox security tests, such as unit-, and system testing

Figure 4: Secure Scrum’s alignment to Process Areas of SSE-CMM

REFERENCES

[1] Agile Alliance. Invest. https://www.agilealliance.org/glossary/invest/
[2] Agile Alliance. User Stories. https://www.agilealliance.org/glossary/user-stories/
[3] Vishal Asthana, Izar Tarandach, Niall O’Donoghue, Bryan Sullivan, and Mikko Saario. 2012. Practical Security Stories and Security Tasks for Agile Development Environments. https://www.safecode.org/publication/SAFECode_Dev_Practices0211.pdf
[4] Barry Boehm and Richard Turner. 2005. Management challenges to implementing agile processes in traditional development organizations. *IEEE software* 22, 5 (2005), 30–39.
[5] Michael Howard and Steve Lipner. 2006. *The security development lifecycle: SDL, a process for developing demonstrably more secure software.* Microsoft Press.
[6] SCRUM institute. 2002. Scrum Release Planning. (2002). http://www.scrum-institute.org/Release_Planning.php
[7] ISO/IEC 21827:2008(en) 2008. Information technology – Security techniques – Systems Security Engineering – Capability Maturity Model (SSE–CMM). (2008).
[8] Hossein Keramati and Seyed-Hassan Mirian-Hosseinabadi. 2008. Integrating software development security activities with agile methodologies. In *2008 IEEE/ACS International Conference on Computer Systems and Applications.* IEEE, 749–754.
[9] Richard Kissel, Kevin Stine, Matthew Scholl, Hart Rossman, Jim Fahlsing, and Jessica Gulick. 2008. Security Considerations in the System Development Life Cycle. *NIST SP 800-64 revision 2* (2008). DOI: http://dx.doi.org/10.6028/NIST.SP.800-64r2
[10] Gary McGraw. 2006. *Software Security: Building Security In.* Vol. 1. Addison-Wesley Professional.
[11] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, and Anandha Murukan. *Improving Web Application Security: Threats and Countermeasures.* Microsoft Press.
[12] Daniel Mellado, Eduardo Fernández-Medina, and Mario Piattini. 2007. A Common Criteria Based Security Requirements Engineering Process for the Development

of Secure Information Systems. *Computer Standards & Interfaces* 29, 2 (2007), 244–253. DOI: http://dx.doi.org/10.1016/j.csi.2006.04.002

[13] Microsoft. 2012. SDL for Agile. (2012). https://www.microsoft.com/en-us/SDL/Discover/sdlagile.aspx

[14] NIST. 2016. National Vulnerability Database. https://nvd.nist.gov/

[15] OWASP. 2013. OWASP Top Ten Project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[16] OWASP. 2015. Application Threat Modeling. https://www.owasp.org/index.php/Application_Threat_Modeling

[17] Christoph Pohl and Hans-Joachim Hof. 2015. Secure Scrum: Development of Secure Software with Scrum. *arXiv preprint arXiv:1507.02992* (2015).

[18] Ken Schwaber and Jeff Sutherland. The Scrum Guide. http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf

[19] Tuuli Siiskonen, Camillo Särs, Antti Väh-Sipilä, and Ari Pietikäinen. 2014. Generic Security User Stories. In *Handbook of the Secure Agile Software Development Life Cycle*, Pietikinen Pekka and Rning Juha (Eds.). University of Oulu, Oulu, Chapter 2, 9–14.

[20] Kevin Stine, Rich Kissel, William C. Barker, Jim Fahlsing, and Gulick Jessica. 2008. Volume 1: Guide for Mapping Types of Information and Information Systems to Security Categories. *NIST SP 800-60 Volume 1 revision 1* (2008). DOI: http://dx.doi.org/10.6028/NIST.SP.800-60v1r1

[21] Antti Vähä-Sipilä. 2010. Product Security Risk Management in Agile Product Management. https://owasp.org/images/c/c6/OWASP_AppSec_Research_2010_Agile_Prod_Sec_Mgmt_by_Vaha-Sipila.pdf

[22] Jaana Wäyrynen, Marine Bodén, and Gustav Boström. Security Engineering and eXtreme Programming: An Impossible Marriage?. In *Extreme Programming and Agile Methods - XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods* (2004-08-15), Carmen Zannier, Hakan Erdogmus, and Lowell Lindstrom (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 117–128. DOI: http://dx.doi.org/10.1007/978-3-540-27777-4_12