Secure Scrum and OpenSAMM for Secure Software Development

Christoph Pohl
and Hans-Joachim Hof

MuSe - Munich IT Security Research Group
Munich University of Applied Sciences
Email: christoph.pohl0@hm.edu, hof@hm.edu

Abstract—**Recent years saw serious attacks on software, e.g., the Heartbleed attack. Improving software security should be a main concern in all software development projects. Currently, Scrum is a popular agile software development method, used all around companies and universities. However, addressing IT security in Scrum projects is different to traditional security planning, which usually requires detailed planning in an initial planning phase. After this planning phase, only minor adjustments are expected. In contrast, Scrum is known for very little initial planning and for constant changes. This paper presents Secure Scrum, an extension to Scrum, that deals with the characteristics of security planning in Scrum. Secure Scrum is a variation of the Scrum framework that puts an emphasis on implementation of security related issues without the need of changing the underlying Scrum process or influencing team dynamics. To implement Secure Scrum in an organization, it helps to utilize a framework for strategic security planning. This paper uses the example of the OpenSAMM (Open Software Assurance Maturity Model) to show how Secure Scrum could be implemented in the field. A field test of Secure Scrum shows that the security level of software developed using Secure Scrum is higher then the security level of software developed using standard Scrum and that Secure Scrum is even suitable for use by non-security experts.**

Keywords–Scrum; Secure Scrum; Secure Software Development; SDL; OpenSAMM.

I. Introduction

This paper presents Secure Scrum and how Secure Scrum can be used in conjunction with OpenSAMM for the development of secure software. Secure Scrum was first presented in [1].

In times of the Internet of Things, even refrigerators now have network support and run a whole bunch of software. As software is so ubiquitous today, software bugs that lead to successful attacks on software systems are becoming a major hassle, see, e.g., [2]. To deal with the constant presence of attacks on systems, modern software development should focus on developing SECURE software, meaning software with little or no vulnerabilities.

Scrum [3][4] is a very popular software develpoment framework at the moment [5]. Unfortunately, Scrum comes without security support. This paper presents Secure Scrum, an extension of the Scrum framework that supports developers in implementing secure software. Secure Scrum is even suitable for non-security experts.

Scrum groups developer in small developer team, which have a certain autonomy to develop software. It is assumed in Scrum that all developers can implement all tasks at hand. Software development projects are split into so-called sprints. A sprint is a fixed period of time (between 2 and 4 weeks). During a sprint, the team develops an increment of the current software version, typically including a defined number of new features or functionality, which are described as user stories. User stories are used in Scrum to document requirements for a software project. All user stories are stored in the Product Backlog. During the planning of a sprint, user stories from the Product Backlog are divided into tasks. These tasks are stored in the Sprint Backlog. A so-called Product Owner is the single point of communication between customer and developer team. Regular feedback of customers on the state of the current increment of the software introduces agility to software development. Changes of user stories reflects this agility. The Product Owner also prioritizes the features to implement. Traditional Scrum does not include any security-specific parts.

One major driver of software security in Secure Scrum is the identification of security relevant parts of a software project. The identification of security critical system parts is very important in any software project, because only in this case, developers can implement appropriate security controls. Traditional software development processes typically use methods of security requirements engineering to identify security critical components of a system. However, the planning moments of Scrum (Product Backlog Refinement, Sprint Planning, and Sprint Review) have very tight time constraints, hence it is very hard to apply time-consuming traditional security requirements engineering methods. In Secure Scrum, security relevance of parts of the emerging software is visible to all team members at all times. This approach is considered to increase the security level, because developers place their focus on things that they had evaluated themselves, which they fully understand, and when their prioritization of requirements does not differ from prioritization of others [6][7].

Secure Scrum aims on achieving an appropriate security level for a given software project. The term "appropriate" was chosen to avoid costly over engineering of IT security in software projects. The definition of an appropriate security level is the crucial point in resource efficient software development (e.g., time and money are important resources during software development). For the definition of an appropriate security level, Secure Scrum relies on the definition in [8]: Software needs to be secured until it is no longer profitable for an intruder to find and exploit a vulnerability. This means that an appropriate security level is reached once the cost of an attack is higher then the expected gain of the attack. So, Secure Scrum offers a way to not only identify security relevant parts of the project but to also judge on the attractiveness of attack vectors in the sense of ease of exploitation.

The identification of security issues is not the only important part of achieving software security, the developers also need to implement effective controls to avoid potential security risks. In Scrum, each team member is responsible for the completeness of his solution (Definition of Done). However, there is a huge number of choices of methodologies to verify completeness. Thus, Secure Scrum must be able to integrate different verification methods. This leads to the issue that Secure Scrum needs to support team members with verification, but without the use of predefined verification methods. This means that a team member can use any method for verification (same as with normal tests, Scrum does not tell the developer how to test). Secure Scrum helps developers to identify appropriate security testing means for security relevant parts of a software project.

One last challenge solved by Secure Scrum is the availability of security knowledge when needed. In standard Scrum, each team member is responsible for his own work, this also means that the team member needs the knowledge to solve the requested task. Nowadays, the availability of security knowledge and experience among software developers does not reflect the importance of this issue. To keep many benefits of standard Scrum, Secure Scrum assumes that the vast majority of requirements should and could be handled by the team itself. However, for some security related issues, it could be necessary or more cost effective to include external resources like security consultants or in-house security experts in the project. Secure Scrum offers a way to include these external resources into the project without breaking the characteristics of Scrum and with little overhead in administration.

The rest of this paper is structured as follows: The following section summarizes related work on software security relevant for Secure Scrum. Section III shows the design of Secure Scrum in detail. Section IV shows how Secure Scrum can be implemented in an arbitrary organization using the framework OpenSAMM. Secure Scrum is evaluated in a field test in Section V. Section VI summarizes the findings of this paper.

II. Related Work

There are several methods for achieving software security, e.g., Clean Room [9], Correctness by Construction [10], CMMI-DEV [11][12], etc. However, these methods cannot be used in Scrum as they do not blend well with the characteristics of agile software development and specifically Scrum. Correctness by Construction [10], for example, advocates formal development in planning, verification and testing. This is completely different to agility and flexible approaches like agile methodologies. Especially Scrum has a strong focus on fast changes to running code, the overhead of Correctness by Construction would be significant. Other models like CMMI-DEV [11][12] can deal with agile methods. The main difference is that CMMI focuses on processes and Scrum on the developers [12]. This means that Scrum and other agile methodologies are developer centric, while CMMI is more process oriented. Restricting developers by rigid processes would break the idea of self-organization of Scrum, hence would introduce significant overhead. Concepts like Microsoft SDL [13] are designed to integrate agile methodologies, but is also self-contained. It can not be plugged into Scrum or any other agile methodology. Scrum focuses on rich communication, self-organisation, and collaboration between the involved project members. This conflicts with formalistic and rigid concepts.

To sum it up, the major challenge of addressing software security in Scrum is not to conflict with the agility aspect of Scrum.

S-Scrum [14] is a “security enhanced version of Scrum”. It modifies the Scrum process by inserting so-called spikes. A spike contains analysis, design and verification related to security concerns. Further, requirements engineering (RE) in story gathering takes effect on this process. For this, the authors describe to use tools like Misuse Stories [15]. This approach is very formalistic and needs lot of changes to standard Scrum, hence hinders deployment in environments already using Scrum. Secure Scrum in contrast is compatible with standard Scrum, hence can be used in environments where Scrum is already used.

Another approach is described in [16]. It introduces a Security Backlog beside the Product Backlog and Sprint Backlog. Together with this artifact, they introduce a new role. The security master should be responsible for this new Backlog. This approach introduces an expert, describes the security aware parts in the backlog, and is adapted to the Scrum process. However, it lacks flexibility (as described in the introduction) and does not fit naturally in a grown Scrum team. Also, the introduction of a new role changes the management of projects. With this approach, it is not possible to interconnect standard Scrum user stories with the introduced security related stories. Secure Scrum in contrast keeps the connect between security issues and user stories of the Product Backlog respectively tasks of the Sprint Backlog.

In [17] an informal game (Protection Poker) is used to estimate security risks to explain security requirements to the developer team. The related case study shows that this is a possible way to integrate security awareness into Scrum. It solves the problem of requirements engineering with focus on software security. However, it does not provide a solution for the implementation and verification phase of software development, hence it is incomplete. Especially, Protection Poker does not ensure that security considerations actually affect the code itself, which is of crucial importance [18]. Secure Scrum in contrast provides a solution for all phases of software development, especially for the important implementation phase.

Another approach is discussed in [19]. An XP Team is accompanied by a security engineer. This should help to identify critical parts in the development process. Results are documented using abuse stories. This is similar to the definition in [20]. This approach is suitable for XP-Teams but not for Scrum.

To sum it up, none of the related work mentioned above integrates well into Scrum, comes with little overhead for Scrum, allows for easy adaption for teams already using standard Scrum, and focuses on all phases of software development. Secure Scrum in contrast solves all of these problems. The design of Secure Scrum is described in detail in the following.

III. Design of Secure Scrum

Secure Scrum consists of four components. These four components are put on top of the standard Scrum framework. Secure Scrum influences six stages of standard Scrum as can be seen in Figure 1.

The components of Secure Scrum are:

- *Identification component*: The identification component is used to identify security issues during software development. To make security issues visible to the team, security issues are marked in the Product Backlog of Scrum. The identification component is used during the initial creation of the Product Backlog as well as during Product Backlog Refinement, Sprint Planning, and Sprint Review.
- *Implementation component*: The implementation component raises the awareness of the Scrum team for security issues during each sprint. The implementation component is used in Sprint Planning, as well as during the Daily Scrum meetings. Hence, all software developers are aware of software security issues all the time.
- *Verification component*: The verification component ensures that team members are able to test the software with focus on the non-functional requirement software security. The verification component gets managed within the Daily Scrum meeting.
- *Definition of Done component*: The Definition of Done component enables the developers to define the Definition of Done for security related parts of the software in a way compatible with standard Scrum. The verification component especially addresses the problem of long-running security tests, e.g., penetration tests, that could not be performed at the end of a Scrum sprint.

In the following, each component of Secure Scrum is described in detail.

A. Identification Component

The identification component is used to identify and mark security relevant user stories. It is used during the initial creation of the Product Backlog as well as during Product Backlog Refinement, Sprint Planning, and Sprint Review. As Product Backlog Refinement, Sprint Planning, and Sprint Review have a very tight time constraint, the identification component does not use traditional methods of security requirements engineering.

Secure Scrum takes a value-oriented approach to security: Software needs to be secured until it is no longer profitable for an intruder to find and exploit a vulnerability. This means that an appropriate security level is reached once the cost of an attack is higher then the expected gain of the attack. Secure Scrum focuses security implementation effort on parts of the emerging software that are of high value for the stakeholders. Hence, in a first step, stakeholders (may be represented by the Product Owner) and team members rank the different user stories according to their loss value. The loss value of a user story is not the cost of development neither the benefit of the functionality that implements the user story. The loss value of a user story is the loss that may occur whenever the functionality that implements the user story gets attacked or data processed by this functionality gets stolen or manipulated. For example, one can formulate “Whenever someone will get access to these data, our company will have high damage”. Even better the cost gets listed with a numerable value like USD or Euro. However, such money estimates tend to be imprecise.

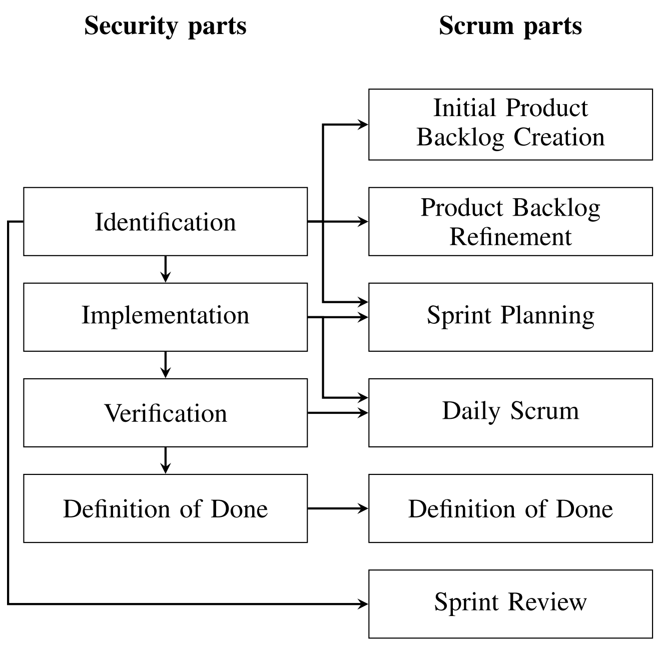

Figure 1. Integration of Secure Scrum components into standard Scrum

In a next step, stakeholders and team members evaluate misuse cases and rank them by their risk. At this point, it can be useful to incorporate external security expertise to moderate by asking the right questions and proposing security aware user stories.

If an organization often develops software for the same domain (e.g., financial service sector, medical sector), it is advisable to compile a list of misuse cases from prior projects in the same domain that could be used for future projects. Also, checklists may be used. Other useful sources for risk estimation are other risk rating methodologies, e.g., the OWASP Risk Rating [21].

After using the identification component, team members and stakeholders have a common understanding of security risks in the Product Backlog. To keep awareness for security risks at a high level, the initial understanding about security risks is documented in the Product Backlog. To do this, Scrum uses so-called *S-Tag*s. Figure 2 shows the basic principle of an *S-Tag*. An *S-Tag* consists of one or more *S-Mark*s, a Backlog artifact, and a connection between the Backlog artifact and one or more *S-Tag*s. An *S-Tag* identifies Product Backlog items that have security relevance with a marker called *S-Mark*. This ensures that the security relevance of certain items in the Product Backlog is visible at all times. The technology

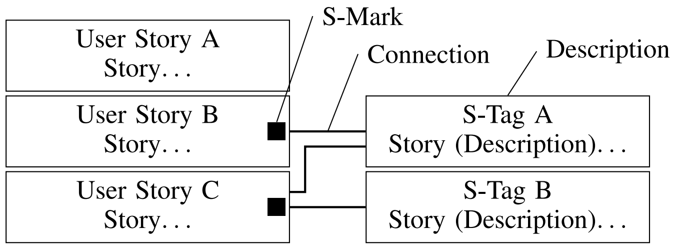

Figure 2. Usage of S-Tags to mark user stories in the Product Backlog and to connect user stories to descriptions of security related issues.

behind the *S-Mark* is negligible (it can be a red background, a dot, or something else), it only must be ensured that a Product Backlog item with security relevance contrasts to other Backlog items.

An *S-Tag* describes one security concern. A detailed description of the security issue helps the Scrum team to understand the security concern. The description of the security concern itself can be formulated in a separate Backlog item. This can be a user story, misuse story, abuse story, or whatever a team decides to use as description technology. The description may include elements from a knowledge base that gives advice on how to deal with this specific security concern. If such a knowledge base is maintained over the course of several projects, it is very likely a valuable source of information for the Scrum team. A knowledge base could also increase the time-efficiency of the identification component.

An *S-Tag* links one security concern to one or more Backlog items. A security concern is any security related problem, attack vector, task, or security principle that should be considered during implementation. One-to-many-connections between security concern and affected Product Backlog items allow for grouping of items that share the same security concern (and hopefully may use the same security mechanisms) as well as expressing security on a high level. Connections between *S-Tag*s could be realized by using unique identifiers for *S-Tag*s that are part of the *S-Mark* (e.g., written on a red dot that is used as *S-Mark*). Using meaningful identifiers helps in understanding security concerns at one glance.

B. Implementation Component

Original Scrum has a strong focus on implementation and running code. Hence, it is obvious that security efforts must affect the code itself [18]. Thus, Secure Scrum makes security concerns visible for the developers at all time. To ensure that security concerns are visible in daily work of the developers, they must be present in the Sprint Backlog. Subsection III-A describes how security concerns are included in the Product Backlog. The Product Backlog lists the required functionalitities of the product. This includes the *S-Tag*s. Usually, a sprint implements a subset of these functionalities (for example user stories). During a sprint, some user stories are broke down to tasks (or similar conceptual parts). Whenever a user story is marked with an *S-Mark*, the corresponding *S-Tag* must also be present in the corresponding Sprint Backlog and the *S-Tag* must be handled by the developer during the sprint. An *S-Tag* can be handled like any other Backlog item. But whenever an *S-Tag* gets split into tasks, these tasks must also be marked with an *S-Mark* and connected to the original *S-Tag*. This ensures that developers are always aware of the original security concern and the security concern can be linked back to the origin description. Using the implementation component of Secure Scrum ensures that developers are aware of the relevant security concerns of the product in each sprint and that security concerns do not get lost during implementation.

C. Verification Component and Definition of Done Component

Increased awareness for security related concerns is not the only advantage of the use of *S-Tag*s. *S-Tag*s are also very useful when identifying requirements for verification of the emerging software. In the first place, *S-Tag*s clearly identify parts of the emerging software that need security verification. In the second place, *S-Tag*s are useful to estimate the effort needed for verification. Some security verifications may need a long time (e.g., penetration testing), hence could not be performed at the end of a sprint.

For further simplification, the term “task” is used for some work that is performed by one developer in one sprint and that needs one Definition of Done.

Secure Scrum proposes two different approaches for verification to deal with the problem of long running tests and, therefore, two variants of the Definition of Done exist:

- *Same-Sprint-Verification:* Whenever the verification process (whatever the developer or team chooses to use) for one task can be performed during the same sprint and by the same developer, the verification must be part of the task itself. This ensures that the verification is also part of the Definition of Done.
- *Spin-off Verification Task:* This variant is used if a developer does not have the required knowledge for verification, or the verification needs external resources, extra time for testing, or anything else that hinders an immediate verification. In this case, the verification cannot be part of the Definition of Done. In such cases, a new task must be created that inherits only the verification part of the original task. This new task (”spin-off verification task”) must be marked with an *S-Mark* and should be connected to the original *S-Tag*, together with the original task. In this case, the developer can define the Definition of Done without the verification, hence a Definition of Done compatible to standard Scrum is available. It is of crucial importance that spin-off verification tasks are subject of sprints in the near future. However, if external experts are used for certain verification tasks (see Subsection III-D), it is beneficial if spin-off verification tasks can be pooled. In any case, it must be ensured that there are no unhandled spin-off verification tasks at the end of software development.

The proposed approach for the definition of the Definition of Done ensures that the connection between an *S-Mark* and its corresponding *S-Tag* keep existing throughout the project, hence no security concern can get lost.

D. Integration of External Ressources

IT security knowledge may be rare in a Scrum team or special knowledge not present in the Scrum team may be necessary for certain parts of the emerging software (e.g.,

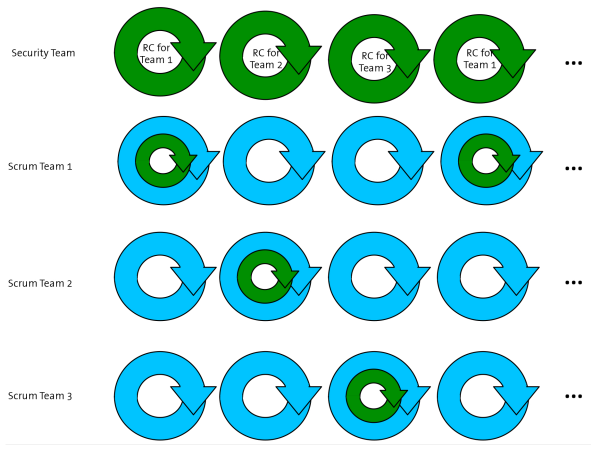

Figure 3. Running Coach approach: one security team provides security services to multiple scrum teams

implementation and testing of cryptographic algorithms, penetration testing of software increments). Or a company prefers to pool security experts in a special team that supports multiple other scrum teams. Such a security team allows for a running coach approach: the security team supports exactly one team at each scrum sprint. If sprints are synchronized in a company, this helps to have a good use of the security experts. Figure 3 shows this so-called running coach approach with one security team that supports three Secure Scrum teams.

Secure Scrum offers ways to include external resources (e.g., external security consultants or internal running coaches) in all components of Secure Scrum. External resources could have one or more of the following three functions:

- Enhance knowledge and provide guidance
- Solve challenges
- Provide external view

These three functions are described in the following.

Enhance knowledge and provide guidance: This function includes security-related training for the Scrum team to help them to gain a better understanding of a specific security-related area. Doing so on the job during a project offers a chance to teach IT security with a specific example at hand (e.g., a certain *S-Tag* that is linked to many user stories) and may be more efficient than security training between two projects. Training may be necessary for aspects that are not part of everyday work, e.g., the usability of security mechanisms [22], [23].

Solving Challenges: Some *S-Tag*s represent hard security challenges that require special expertise or special experience, such that it is more cost efficient to let external resources solve this challenge. To avoid breaches in Scrum, it is necessary that these external solutions can be handled like a tool, a well defined part of development, a framework, or a “black box”, which is ready to use. This means that this external solution should be encapsulated and therefore does not influence Scrum or the Scrum team. For example, this can be a functional part of software (with special IT Security concerns) or parts of the project, which can be used with an API by the Scrum team. Another challenge is the integration of external services like penetration testing into the development process. One way to do so is that external resources provide test cases (e.g., for Metasploit [24]) that can be used for every increment of the emerging software at any time. Results of tests can be documented as artifacts in the Backlog. Then they can be handled like any other change request.

Providing external view: One major part in IT Security is to recognize ways to exploit the own system. In other words, one must think like an attacker to recognize potential attack vectors. Usually, it is easier for an outsider to spot potential weaknesses of a system than it is for the developer of a system. Hence, external resources may introduce a valuable external viewpoint on a project. When using the identification component of Secure Scrum, an external consultant can be helpful to point the team to security concerns. When using the implementation component, external resources can be helpful in the sprint planning or could perform code reviews. When using the verification component, an external consultant can help to create tests for security concerns. These interventions by external resources should not be part of the normal Scrum processes, the external resource should only help to ask questions (in the meaning of: he should show relevant concerns in scope of IT Security). In conclusion, the external resource should help to set focus on problems the team is not aware of.

IV. Implementation of Secure Scrum Using OpenSAMM

This section gives some hints on how to successfully implement Secure Scrum in an organization. Implementation of a new security approach is a non-trivial task for many organizations. Secure Scrum is compatible with many frameworks for implementation of security strategies, and if there is already a framework in use in an organization, it is a good idea to use this framework to implement Secure Scrum. Existing frameworks may be for example Microsoft SDL [25] or CLASP (Comprehensive Lightweight Application Security Process) [26]. This section uses OpenSAMM (Open Software Assurance Maturity Model) [27] as an example to show possible implementation activities. OpenSAMM was chosen because it can be used with arbitrary security strategies, has a small overhead, is open and flexible enough to be a good fit for security in agile software development. One big advantage of OpenSAMM compared to Micrsoft SDL, CLASP, and many other frameworks is, that it it is not necessary to introduce Secure Scrum in one big project at one time, but many small steps are possible, using so-called maturity levels. Especially this feature of OpenSAMM makes it a good fit for small and medium size companies with small security budgets.

OpenSAMM consists of four business functions that are typical for organizations developing software. Each business function has three security practices (see Figures 5, 6, 7, and 8). Each security practices has levels between 0 (no security yet) and 3 (mature security). The meaning of these maturity levels is described in the following:

- Maturity level 0: No activities for this security practice are implemented. In most organizations, this is the starting point.
- Maturity level 1: There is a basic understanding of the security practice and first implementations of the security practice exist.

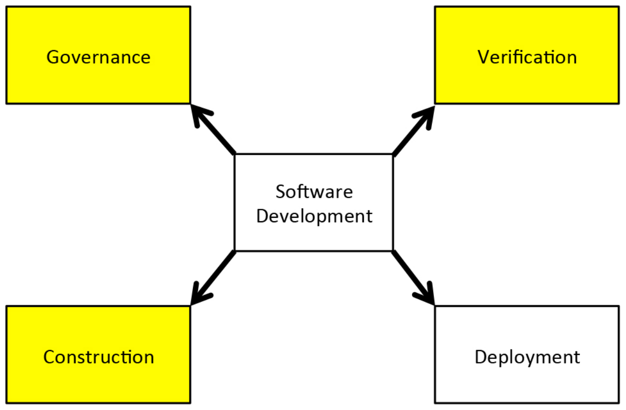

Figure 4. Overview OpenSAMM (parts relevant for the implementation of Secure Scrum in yellow)

- Maturity level 2: Efficiency and/or effectiveness of implementations of this security practice get enhanced at this level.
- Maturity level 3: The security practice is at a high level of competence.

This section describes security activities for Secure Scrum security practices on levels 1 through 3. The level approach helps to introduce Secure Scrum in multiple steps. OpenSAMM includes methodologies to verify the successful implementation of activities on a certain level of a security practice before proceeding further. Also, OpenSAMM offers roadmap templates for typical domains. Figure 4 gives an overview of the four business functions of OpenSAMM. The four business functions of the software development process in OpenSAMM are:

- *Governance*: The focus of the business function Governance lies on the overall processes and activities for software development in an organization.
- *Construction*: The focus of the business function Construction lies on how to create software in a software project. This business function is the most important business function for the implementation of Secure Scrum.
- *Verification*: The focus of the business function Verification lies on how to test software produced during software development. Typical activities include penetration testing, general software quality assurance actions as well as manual review of source code or even design documents.
- *Deployment*: The focus of the business function Deployment lies on how to manage releases of software. This includes shipping of products to the end user, installation of products as well as operational aspects of software. There is no need to adapt this business function for Secure Scrum as operation of software is out of scope of Secure Scrum.

Secure Scrum needs to be included in the business functions Governance, Verification, and Construction. The business function Deployment is out of scope of Secure Scrum, but nevertheless, it is very important for achieving software security in general.

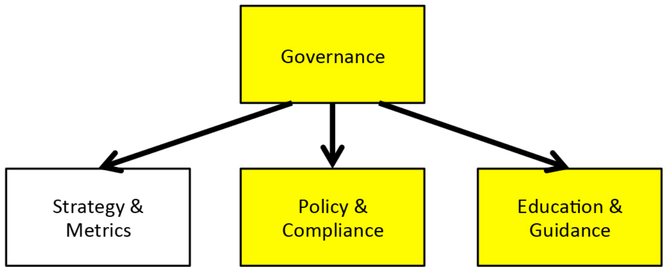

Figure 5. Security practices of business function Governance (parts relevant for the implementation of Secure Scrum in yellow)

Figure 5 shows the security practices for the business function Governance. They are:

- *Strategy & Metrics*: This security practice includes the overall strategy for development of secure software as well as metrics to measure progress in enhancing the security level of software.
- *Policy & Compliance*: This security practice includes setting up control mechanisms to check that all processes for development of secure software are followed.
- *Education & Guidance*: This security practice includes all activities that enhance knowledge of software developers.

Security practice Strategy & Metrics does not need changes for Secure Scrum. Secure Scrum can be used in many different security strategies and all kind of metrics can be used. See [28] for an overview of common metrics.

Security practice Policy & Compliance includes Secure Scrum activities at the following maturity levels:

- Maturity Level 1: Establish compliance guidelines for Secure Scrum usage. Guidelines should include mandatory use of Secure Scrum as well as responsibilities of the Scrum roles (process owner, Scrum Master, team members) in Secure Scrum, e.g., who is responsible for risk rating.
- Maturity Level 2: Establish compliance guidelines for risk rating in software projects. Establish regular audits of projects.
- Maturity Level 3: Establish a solution for audit data collection. Establish compliance gates, e.g., check compliance with Secure Scrum guidelines every 6th sprint.

Security practice Education & Guidance includes Secure Scrum activities at the following maturity levels:

- Maturity Level 1: Establish a technical guideline for Secure Scrum. This should include a description of the Secure Scrum components Identification, Implementation, Verification, and Definition of Done components (see Section III), a comprehensive description of S-Marks and S-Tags as well as a description of the integration of Secure Scrum in Scrum. Establish a knowledge base of security concerns as described in Section III-A.

- Maturity Level 2: Each scrum role (product owner, scrum master, scrum team member) gets role-specific security training, an introduction to Secure Scrum as a methodology as well as an introduction to the supporting systems. Project teams are supported by Secure Scrum running coaches to support the transition to Secure Scrum.
- Maturity Level 3: Establish mandatory Secure Scrum training for all roles. Establish a role-based Secure Scrum exam or a role-based Secure Scrum certification.

Figure 6 shows the security practices for the business function Construction. They are:

- *Threat Assessment*: Identification of threats and risk assessment.
- *Security Requirements*: Specification of security requirements.
- *Secure Architecture*: Activities to achieve a secure software design.

All three security practices need to include Secure Scrum related activities.

Security practice Threat Assessment includes Secure Scrum activities at the following maturity levels:

- Maturity Level 1: Establish a knowledge base of application-specific typical attacks for use in the identification component of Secure Scrum.
- Maturity Level 2: Adopt a system for rating relevant attacks per application. Such a system could for example be based on the OWASP Risk Rating [21]. Establish a knowledge base of misuse-cases that are typical for the developed applications.
- Maturity Level 3: Include external experts in risk assessment, see Section III-D for details.

Security practice Security Requirements includes Secure Scrum activities at the following maturity levels:

- Maturity Level 1: Establish mandatory use of the identification component of Secure Scrum. Use knowledge base in identification of relevant attacks. Connect affected user stories with knowledge base articles.
- Maturity Level 2: Establish a risk-based approach for the identification component.
- Maturity Level 3: Audit S-Mark usage explicitly during software development. Audit use of spin-off verification tasks. Ensure no spin-off verification tasks exist at the end of software development.

Security practice Secure Architecture includes Secure Scrum activities at the following maturity levels:

- Maturity Level 1: Maintain a list of security design principles in a knowledge base and make sure that S-Marks for each product are connected to them.
- Maturity Level 2: Maintain a list of security design patterns [29] and make sure that S-Marks are connected to them.
- Maturity Level 3: Maintain a list of reference architectures and make sure that S-Marks connect to them. Use audits to ensure that secure frameworks, patterns, and platforms are used. Establish regular audits by external security experts.

Figure 7 shows the security practices for the business function Verification. They are:

- *Design Review*: Inspection of the design regarding the use of adequate security mechanisms
- *Code Review*: Inspection of code to find potential vulnerabilities.
- *Security Testing*: Testing of software increments produced during software development for vulnerabilities.

All three security practices need to include Secure Scrum related activities.

Security practice Design Review includes Secure Scrum activities at the following maturity levels:

- Maturity Level 1: Establish mandatory use of S-Marks at the start of a project to tag security relevant user stories.
- Maturity Level 2: Establish audit of all S-Marks by external security experts.
- Maturity Level 3: Establish periodic audits (e.g., every 6th Scrum sprint) by all roles of Scrum as well as by external security experts to review the user stories and assign S-Marks.

Security practice Code Review includes Secure Scrum activities at the following maturity levels:

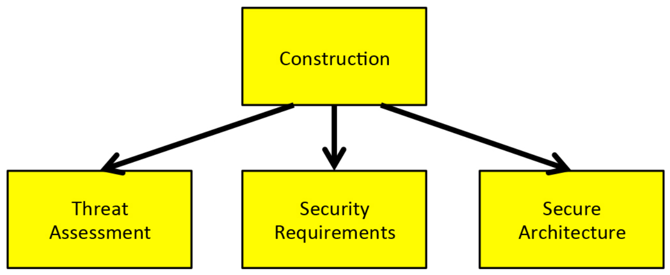

Figure 6. Security practices of business function Construction (parts relevant for the implementation of Secure Scrum in yellow)

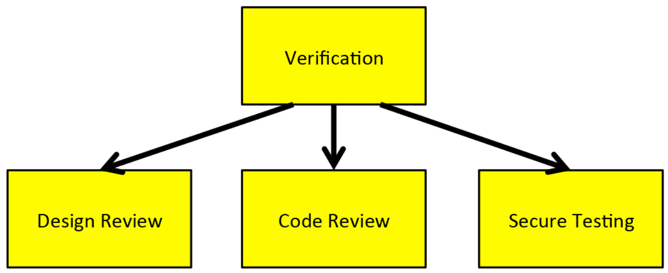

Figure 7. Security practices of business function Verification (parts relevant for the implementation of Secure Scrum in yellow)

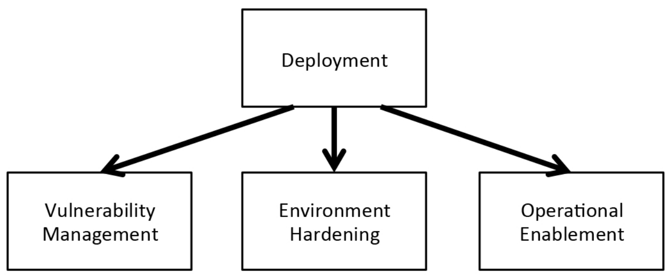

Figure 8. Security practices of business function Deployment (parts relevant for the implementation of Secure Scrum in yellow)

- Maturity Level 1: Establish spin-off verification gates: At certain points during the software development make sure, that spin-off verification task in backlogs are cared for.
- Maturity Level 2: Offer support of security experts to verify verification artifacts.
- Maturity Level 3: Use experts to verify all verification artifacts (e.g., running coaches, see Section III-D, or external penetration testers).

Figure 8 shows the security practices for the business function Deployment. They are:

- *Vulnerability Management*: Establish processes to handle vulnerability reports.
- *Environment Hardening*: Increase security level of systems that run software provided by the organization
- *Operational Enablement*: Provide security information to administrators that use the organizations software. This information includes secure configuration, secure deployment, and security operation.

None of these security practices is relevant to Secure Scrum. However, the three security practices of the business function Deployment are very important for software security in general, hence should hold activities appropriate for the implementing organization.

The OpenSAMM Guide [27] offers several roadmap templates, giving a schedule for security practice level changes. In most cases, the roadmap template "Independent Software Vendor" is a good fit.

V. Evaluation

The evaluation presented in this paper focuses on the following questions:

- Is Secure scrum a practicable approach to develop secure software?
- Is Secure Scrum easy to understand? Does Secure Scrum raise the complexity for applying Scrum?
- Does Secure Scrum increase the security level of the developed software?

For a test setting, 16 developers were asked to develop a small piece of software. The developers were third year students in a computer sciences and business informatics (BSc) study program. They were not aware that they are part of this evaluation. The students showed programming skills that were on the usual level of a third year bachelor student. No participant attended a specialized course in IT Security before beside the compulsory lecture in IT Security (basic level) in the second year of the bachelor. When asked about their practical experience in IT security, all students said that they have no practical experience. Hence, it is expected that none of the students has IT security as a hobby and all students are on the same level concerning IT security knowledge and experience. All developers had average theoretical knowledge about Scrum. Only two students had practical experiences (less than 2 months) with Scrum.

The developers were divided into three groups:

1) Team 1 (T1): The Anarchist group: They could manage themselves as they like, except using Scrum.
2) Team 2 (T2): The Scrum group: They should use standard Scrum.
3) Team 3 (T3): The Secure Scrum group: They should use Secure Scrum.

To avoid influences on the evaluation, teams 1 and 2 thought that team 3 also uses standard Scrum. All groups got a list of six basic requirements for a new software product. They were asked to develop a prototype for a social network with the following features:

- registration,
- login and logout,
- personal messages,
- wall messages,
- bans, and
- friend lists.

Each group had only one week to develop a prototype of this application using Java and a preconfigured spring framework template (based on BREW (Breakable Web Application) [30]). Each group was asked to implement as many requirements as possible. However, it was known that it is impossible to implement all requirements for the final version of the application considering the harsh time constraints. This setting of the evaluation assures a high time pressure (as in real projects), hence allows to observe the prioritization of security-related tasks. Developers were also told that they need to "sell" their prototype on the last day of the experiment in front of a jury. In fact they should learn how to present their prototype and act like a team that wants to have a contract for further development. This should ensure that every team needs to define for itself the selling points of their prototype, putting a high pressure on feature richness. Again, this setting helps to evaluate the prioritization of security-related task. Team 3 has a short Secure Scrum briefing of about one hour. Every team is advised to produce a proper documentation. This includes all produced artifacts, the sources, and a short description of their development process.

Table I summarizes some basic findings of the experiment.

All three teams had a rough definition of the six basic requirements, which should be implemented. They were told that whenever the requirements list should be enhanced to deal with the 6 requirements given by the customer, they are free to define new requirements. Team 1 did not define any new requirements. Team 2 defined one new requirement to enhance

TABLE I. Results of the evaluation of the efficiency and effectiveness of Secure Scrum

#	Metric	T1	T2	T3
1	Lines of code	1149	758	458
2	Number of basic requirements	6	6	6
3	Number of additional requirements defined	0	1	8
4	Number of basic requirements documented	0	6	6
5	Number of basic requirements implemented	6	5	4
6	Number of requirements documented	0	7	14
7	Number of requirements implemented	6	6	9
8	Number of vulnerabilities sp	18	12	3
9	Group size	6	5	5

performance. Team 3 defined 8 new requirements that had a focus on IT Security. These requirements are an excerpt of the descriptions for the *S-Tag*. Overall, they defined 29 new stories focused on IT Security. This shows that even with beginner skills in computer sciences and only basic skills in IT Security, it is possible to define a high amount (compared to the original requirements) of security related requirements. It also shows that it is possible to describe the most problematic vulnerabilities or problems with the help of risk identification.

Metrics $4-7$ of Table I are used to evaluate if the teams documented all requirements and how many of the requirements were implemented. The evaluation shows that the teams did not take care of any further requirements when not specified by the customer. This sounds trivial, but it also shows that the developer did not take care of IT Security when not specified. The Secure Scrum team (team 3) is the only team that did not implement all given basic requirements. Instead, they obviously prioritized some of the security requirements over the basic requirements as some of the additional requirements that were added by the team were implemented. This finding shows that Secure Scrum succeeds in putting focus on software security.

Metric 8 shows the number of security problems that were created by the developers. The number of security problem is calculated as follows:

Let sl be a vulnerability listed in the OWASP Top 10 list OTT ($sl \in OTT$). The OWASP Top 10 project [31] lists the most common security vulnerabilities for web applications:

- Injection,
- Broken Authentication and Session Management,
- Cross-Site Scripting (XSS),
- Insecure Direct Object References,
- Security Misconfiguration,
- Sensitive Data Exposure,
- Missing Function Level Access Control,
- Cross-Site Request Forgery,
- Using Components with Known Vulnerabilities, and
- Unvalidated Redirects and Forwards.

Let OS be the complete source code of the developed software and SC the part of the software written by the students ($SC \subset OS$). Let cf be a Java function. Let $cpf(sl)$ be a function that counts the amount of sl for one cf. By definition, $cpf(sl)$ increments a vulnerability counter by one

TABLE II. Results of the evaluation of the practicality of Secure Scrum

#	Metric	Team 2	Team 3
1	Number of requirements	7	14
2	Number of user stories	7 (13)	14 (62)
3	Number of tasks	18	35
4	Number of user stories with *S-Mark*	-	14
5	Number of tasks with *S-Mark*	-	8(35)

whenever the current function is the source ms function for a vulnerability. A function cf is considered as a source ms whenever $cf \in SC$ and when the function is the reason for the vulnerability or it calls a function cf_1 where $cf_1 \notin SC$ and cf_1 is the reason for the vulnerability. The amount of vulnerabilities sp is the sum of all $cpf(cf)$. Such a definition of the number of security problems only counts code that is responsible for vulnerabilities of a software system. It also takes into consideration the use of vulnerable code. For example, when a developer creates an SQL statement with a potential SQL Injection vulnerability, the function holding the database call with this statement is regarded as the reason of the vulnerability.

The results of the evaluation shows that team 1 and team 2 had a high amount of vulnerabilities in their software (team 1: 18, team 2: 12). Both teams built software exploitable by SQL Injection, XSS, CSRF, and had a vulnerable session management. Team 3 had significantly less vulnerabilities. It should be noted that every team has the same level of security knowledge. The benefit of team 3 is, that their usage of Secure Scrum raised the awareness for security issues. Hence, the evaluation shows that the use of Secure Scrum increase the security level of the developed software.

The first metric (Lines of Code (LOC)) of Table I shows the amount of code, which was generated during the week. There are significant differences between the three teams. The teams that identified additional requirements (performance (team 2) and security (team 3)) were not as productive as the other teams. The difference between team 2 and team 3 shows that Secure Scrum raises the complexity of applying Scrum. This shows the overhead that comes with a broadened focus on software quality, especially on non-functional requirements. However, it should be noted that it is expected that the overhead of Secure Scrum decreases over time as team members get used to it and as a knowhow transfer over project takes place, e.g., in the form of knowledge base entries on security issues (see Section III-A) or security checklists from previous projects (see Section III-A).

Secure Scrum is considered to be easy to use and practicable, if even students with a weak background in IT security are able to identify security relevant parts of the software. To evaluate ease of use and practicality of Secure Scrum, the documentation of the Scrum teams was evaluated. The documentation consists of the Backlogs and a timetable. There, it can be seen if the Secure Scrum team did identify security relevant parts of the software.

Table II summarizes the results of this evaluation for team 2 (Scrum) and team 3 (Secure Scrum) to compare standard Scrum to Secure Scrum.

Numbers in braces give the total amount of user stories. The numbers not in braces (aggregated number) show the amount of user stories when grouped together. This means a group of user story is a "bigger" user story, which reflects a requirement. Team 2 broke down every user story to a different task. Team 3 broke down tasks for only the stories that they also implemented. This is why they defined more user stories than tasks. Team 3 found for every user story some security concerns, this is why they tagged all user stories. Metric 5 shows that all tasks also had *S-Mark*s, overall they had 8 different groups in the tasks. Team 3 decided to create the links by grouping, they simply used red cards for the descriptions to show security problems (*S-Mark*). This also shows that the proposed tools are simple enough to adapt them very fast in a Scrum process.

In conclusion, the evaluation shows that Secure Scrum is able to improve the security level of the developed software. Secure Scrum is easy to understand, can be used in practice, and is even suitable for teams that have no deepened security knowledge. The evaluation also shows that it is possible to have a proper documentation through all stages of the experiment. The tools of Secure Scrum harmoniously blend into the standard Scrum toolset without the need of much overhead for training.

VI. Conclusion

This paper presents Secure Scrum, an extension of the software development framework Scrum. Secure Scrum enriches Scrum with features focusing on building secure software. One of the main contributions of Secure Scrum are *S-Tag*s, a way to annotate Backlog items with security related information. Such annotations help software developers to keep security in mind during software development. The paper also presents how OpenSAMM can be used to implement Secure Scrum in an organization. The maturity level approach of OpenSAMM helps to implement Secure Scrum step by step, hence does not overburden organizations. Secure Scrum was evaluated in a small software development project. The evaluation shows that Secure Scrum can be used in practice, is easy to use and understand, and improves the level of software security.

References

[1] C. Pohl and H.-J. Hof, "Secure Scrum: Develpoment of Secure Software with Scrum," in SECURWARE 2015: The Ninth International Conference on Emerging Security Information, Systems and Technologies. Venice, Italy: IARIA XPS Press, 2015, pp. 15–20.

[2] Symantec, "2015 internet security threat report.", retrieved: 05, 2016 [Online]. Available: https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf

[3] K. Beck, M. Beedle, K. Schwaber, and M. Fowler, "Manifesto for agile software development,", retrieved: 05, 2016 [Online]. Available: http://www.agilemanifesto.org/

[4] K. Schwaber, "SCRUM development process," in Business Object Design and Implementation, D. J. Sutherland, C. Casanave, J. Miller, D. P. Patel, and G. Hollowell, Eds. Springer London, pp. 117–134.

[5] VersionOne, "9th Annual State of Agile Survey.", retrieved: 05, 2016 [Online]. Available: http://info.versionone.com/state-of-agile-development-survey-ninth.html

[6] C. Riemenschneider, B. Hardgrave, and F. Davis, "Explaining software developer acceptance of methodologies: a comparison of five theoretical models," IEEE Transactions on Software Engineering, vol. 28, no. 12, Dec. 2002, pp. 1135–1145.

[7] L. Vijayasarathy and D. Turk, "Drivers of agile software development use: Dialectic interplay between benefits and hindrances," Information and Software Technology, vol. 54, no. 2, Feb. 2012, pp. 137–148.

[8] C. Herley, "Security, cybercrime, and scale," Communications of the ACM, vol. 57, no. 9, Sep. 2014, pp. 64–71.

[9] H. D. Mills and R. C. Linger, "Cleanroom Software Engineering: Developing Software Under Statistical Quality Control - Encyclopedia of Software Engineering - Mills - Wiley Online Library," 1991.

[10] A. Hall and R. Chapman, "Correctness by construction: developing a commercial secure system," IEEE Software, vol. 19, no. 1, 2002, pp. 18–25.

[11] M. B. Chrissis, M. Konrad, and S. Shrum, CMMI for Development, ser. Guidelines for Process Integration and Product Improvement. Pearson Education, Mar. 2011.

[12] H. Glazer, J. Dalton, D. Anderson, M. D. Konrad, and S. Shrum, "CMMI or Agile: Why Not Embrace Both!" 2008, pp. 1–48.

[13] M. Howard and S. Lipner, The security development lifecycle. O'Reilly Media, Incorporated, 2009.

[14] D. Mougouei, N. F. Mohd Sani, and M. Moein Almasi, "S-scrum: a secure methodology for agile development of web services." World of Computer Science & Information Technology Journal, vol. 3, no. 1, 2013, pp. 15–19.

[15] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," Requirements Engineering, vol. 10, no. 1, Jan. 2005, pp. 34–44.

[16] Z. Azham, I. Ghani, and N. Ithnin, "Security backlog in scrum security practices," in Software Engineering (MySEC), 2011 5th Malaysian Conference in. IEEE, 2011, pp. 414–417.

[17] L. Williams, A. Meneely, and G. Shipley, "Protection poker: The new software security," IEEE Security & Privacy, no. 3, 2010, pp. 14–20.

[18] S. B. Lipner, "Security Assurance - How can customers tell they are getting it?" Communications of the ACM, vol. 58, no. 11, 2053, pp. 24–26.

[19] G. Boström, J. Wyrynen, M. Bodn, K. Beznosov, and P. Kruchten, "Extending XP practices to support security requirements engineering," in Proceedings of the 2006 international workshop on Software engineering for secure systems. ACM, 2006, pp. 11–18.

[20] J. Peeters, "Agile security requirements engineering," in Symposium on Requirements Engineering for Information Security, 2005.

[21] Open Web Application Security Project (OWASP), "OWASP Risk Rating Methodology." , retrieved: 05, 2016 [Online]. Available: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

[22] H.-J. Hof, "Towards Enhanced Usability of IT Security Mechanisms - How to Design Usable IT Security Mechanisms Using the Example of Email Encryption," International Journal On Advances in Security, vol. 6, no. 1&2, 2013, pp. 78–87.

[23] H. J. Hof, "User-Centric IT Security - How to Design Usable Security Mechanisms," in The Fifth International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services (CENTRIC 2012), 2012, pp. 7–12.

[24] Rapid7, "Metasploit," 2015, retrieved: 05, 2016. [Online]. Available: http://www.metasploit.com/

[25] M. Howard and S. Lipner, The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software (Developer Best Practices), ser. Secure Software. Microsoft Press, Jun. 2006.

[26] Open Web Application Security Project (OWASP), "CLASP (Comprehensive, Lightweight Application Security Process).", retrieved: 05, 2016 [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_CLASP_Project

[27] P. Chandra, "Software assurance maturity model.", retrieved: 05, 2016 [Online]. Available: http://www.opensamm.org/downloads/SAMM-0.8.1-en_US.pdf

[28] N. Fenton and J. Bieman, Software Metrics - A Rigourous and Practical Approach. CRC Press, 2015.

[29] E. Fernandez-Buglioni, Security Patterns in Practice: Designing Secure Architectures Using Software Patterns(Wiley Series in Software Design Patterns). John Wiley & Sons, 2013.

[30] C. Pohl, K. Schlierkamp, and H.-J. Hof, "BREW: A Breakable Web Application for IT-Security Classroom Use," in Proceedings: European Conference on Software Engineering Education 2014. ECSEE, 2014, pp. 191–205.

[31] Open Web Application Security Project (OWASP), “OWASP Top Ten Project.”, retrieved: 05,2016 [Online]. Available: https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013