Managing Security Work in Scrum: Tensions and Challenges

Sven Türpe and Andreas Poller

Fraunhofer SIT, Darmstadt, Germany

Abstract. We advocate a change of perspective in the question of agile secure software development and analyze what makes it difficult to address security needs in Scrum. The literature focuses on the integration of security activities into agile development processes. However, detailed prescriptions for security work would be misplaced in a generic management framework like Scrum. Therefore we take a closer look at the tensions between Scrum's way of organizing work and the characteristics of security requirements. Our previous work suggests that Scrum works well as a management model and security development requires iterations as in agile development, yet Scrum teams can fail to address security needs due to their low visibility, competing objectives, and Scrum's division of labor. Tensions arise as Scrum is optimized to fulfill explicit requirements and maximize business value, whereas security is often an implicit requirement with a different value proposition, which nevertheless requires substantial work and cannot be addressed by bug fixing or quality assurance alone. As a consequence, promising research directions are the reflective discovery of security needs, the valuation and prioritization of security work, collaboration between Scrum teams and security experts, and verification and feedback mechanisms for security.

Keywords: Scrum, security requirements, security work, management, agile development, software security

1 Introduction

When agile software development entered the scene [1], it was met with skepticism from security experts. While some agile practices could be beneficial for security, others contradicted established approaches to security assurance [2,3]. Empirical results suggest that agile development may indeed tend to neglect security [4] as a so-called non-functional or quality requirement [5]. On the other

In: M.G. Jaatun, D.S. Cruzes (eds.): Proceedings of the International Workshop on Secure Software Engineering in DevOps and Agile Development (SecSE 2017), published at http://ceur-ws.org

hand, some of the early skepticism [3] seems to have its roots in traditional conceptions of security assurance tailored to traditional development processes.

Agile frameworks like Scrum [6,7] and a wide range of agile practices [8] are now the mainstream in software development. Software security has simultaneously become ever more important. Several proposals have appeared over the years how to integrate security engineering practices into agile methodologies like Scrum and XP [9,10,11,12,13]. Despite their different approaches, they build on the common premise that agile teams neglect security because it is not an explicit part of common agile frameworks. Consequently, they present and evaluate extensions to the processes, artifacts, and roles of development frameworks to ensure that security requirements get due attention.

However, the most popular agile methodology, Scrum, is a management framework purposefully honed down to the essentials of roles, responsibilities, and collaboration. Scrum does and should not prescribe development work in detail, but rather aims to create an environment where development teams can self-organize and take responsibility for their work while being managed to the extent necessary for a project to succeed. The key to this management scheme is the delicate balance between the respective responsibilities and spheres of relative autonomy of the development team and the product owner. Organized collaboration between these roles, with the scrum master as a facilitator, aims to ensure that development moves in a productive direction at any time, this direction gets adjusted as requirements change or are better understood, and management antipatterns like micromanagement do not occur.

Our own previous work shows two opposite tendencies. On the one hand, security requirements engineering needs iterations, because design decisions change security properties and adversaries may adapt their behavior to available attack opportunities [14]. One might therefore expect agile development to be well suited to identify and address security needs. On the other hand, we observed obstacles to the adoption of security practices by a Scrum team after an intervention and despite the developers' perceived need for more security [15]. This observation can in part be explained with the theory of organizational routines and with challenges of change management. Another, equally important part of the explanation lies in subtle mismatches between Scrum's strengths and the peculiarities of security as a class of requirements. For example, development teams cannot silently take care of security without product management or other stakeholders explicitly motivating and requiring it, but security may remain invisible and intangible for those.

In this paper we synthesize our previous work on security requirements and on security work in a Scrum team. We theorize about tensions between the characteristics of security requirements and security work on the one hand and the way Scrum manages development work on the other. Analyzing the definition of Scrum, we find three different ways of managing security work: as bug fixing on demand, continuously as a quality requirement through the definition of "done," or as prioritized and planned development work through the product backlog. We discuss the capabilities and limitations of these approaches and find each of them

inadequate. On-demand fixing rarely leads to substantial security improvement. As a quality requirement, security has a complex relationship with development work and is difficult to verify. Security features in the backlog would be a suitable approach to many security concerns, but they compete with other requirements and also may need special expertise to design and implement effectively.

We argue that research aiming to reconcile security engineering with agile development should consider not only the execution of security activities in an agile process, but also these challenges of managing security work in agile frameworks. Our analysis suggests four areas of security tasks that are worth investigating and supporting: the reflective discovery of security needs to create backlog items, the valuation and prioritization of security work, agile verification and feedback in the security dimension, and the collaboration of Scrum teams with external security experts and consultants.

2 Background

2.1 Agile Software Development

Agile software development [16,17] as a concept comprises fundamental values and principles [1], frameworks like XP and Scrum, and a wider set of development practices supporting agility [18,8]. Its basic values and principles emphasize the continuous delivery of valuable software and an empirical approach to requirements, which are explored interactively with the customer and are expected to change. Agile development focuses on the software itself as the product of development, avoiding software bureaucracy in favor of face-to-face communication and a minimal number of lightweight artifacts besides product increments. Self-organizing teams of skilled and motivated people are trusted to get their job done without detailed prescriptions through continuous collaboration with their customers. Under this assumption, reflection, learning, and self-improvement are essential parts of agile development. Finally, the principles of the Agile Manifesto also call for technical excellence and simplicity.

Agile frameworks like XP and Scrum implement the values and principles of agile development, specifying management structures, processes, artifacts, and practices. They organize development work in short, incremental iterations with feedback loops and specify team activities focused on communication and coordination. While Scrum is strictly focused on management aspects—roles, interactions, artifacts—XP also prescribes practices like pair programming and unit testing. Around these frameworks a range of practices [18,8] has emerged that are commonly employed by agile teams regardless of which framework they use. Examples are continuous integration and testing with a high degree of automation and the representation of requirements in the form of user stories.

While agile practices have been widely adopted and became the prevalent paradigm of software development, they have also downsides. Agile development pursues conflicting goals difficult to balance, such as continuous delivery of software and continuous learning [19]. Short development cycles can create

iteration pressure, thus inhibiting activities like reflection and learning that are not directly contributing to the intended iteration results [20].

2.2 Security in Agile Development

Security experts tend to be skeptical about the ability of agile development to produce secure software. Some evidence supports this view: security has a strong non-functional, quality component [21,3] and agile development apparently tends to neglect non-functional requirements [4]. However, early skepticism [2,3] had its roots elsewhere. Traditional approaches to security assurance, such as security certification, assumed traditional development practices in traditional organizations. Some alleged problems of security in agile development were therefore really the result of a culture shock, as becomes obvious in statements like this: *"Pair Programming (...) is not possible in organizations in which workstation sharing is not permitted."* [3]. Agile development seems indeed out of place in such an environment. Note also that *security assurance* does not equal *security*, the former focusing on paperwork to convince someone that due attention has been paid to security and specific requirements are fulfilled.

Various proposals and investigations have appeared over time regarding security activities in agile development processes [22]. For example, Boström et al. [9] propose an extension of XP with practices for security requirements engineering. They introduce abuser stories as a new artifact type to represent security requirements and describe activities creating, refining, and using these artifacts. Kongsli [23] made a similar proposal among suggestions about improved testing and dedicated security review meetings, and Mead et al. [24] showed ways to adopt the SQUARE security requirements methodology for agile development.

Ben Othmane et al. [11] take a different approach and propose a way of representing and updating security assurance arguments in an iterative development process. Besides proposals with such comprehensive aims there are also adaptations of specific agile practices to security. Protection poker [10] is an example, using an approach similar to planning poker for risk assessment. Others have suggested integrating single conventional security activities such as risk analysis [25] into agile development methods or adopting portions of secure software development lifecycles like Microsoft SDL, Cigital Touchpoints or CLASP [26,27,28]. Bartsch [29] criticized that this kind of proposals often provides only a theoretical perspective on method compatibility and potential method adoption strategies. He argues for an empirical perspective focusing on actual development practices, the context of agile development, and in particular on customer-developer interactions.

Rindell et al. argue in a recent literature review [22] that the inherent insecurity of agile development, suspected by early work and echoed in the literature ever since, is a myth. While we agree with the gist of their result, our own work [15], outlined in section 4.2 below, suggests that Scrum as one particular agile management framework poses challenges for the management of security work. Hence we take in this paper a closer look at these challenges. Our prior

research highlights the importance of roles, collaboration, and organizational routines, as opposed to artifacts and formal process.

3 Software Security

3.1 Security Requirements

To protect a system or software product against malicious attacks, its developers have to avoid vulnerabilities that attackers might exploit. Vulnerabilities are introduced inadvertently and it takes additional effort to prevent or remove them. They often appear in side-effect behavior, which does not affect normal use and operations but can be triggered by attackers [30]. Most vulnerabilities therefore remain invisible until one specifically looks for them and analyzes the behaviors of a program through an attacker's lens.

Understanding the security needs of a system under development is a problem of requirements engineering, but security needs are not requirements like any other. Security needs are shaped by three interdependent components [14]: the threats to defend against, the security goals of stakeholders, and the design of the system itself. Threats are adaptive and exploit whichever vulnerability serves their objectives. Design and implementation decisions entail security needs. Security goals determine priorities. To understand security requirements, one needs to analyze all three components together.

Due to these interdependent aspects, security requirements cannot be elicited from stakeholders alone. The design and implementation of a development product itself is a major source of security needs. Architectural decisions, technology choices, and implementation detail lead to new or refined security concerns.

The relationships between the three components are complicated. Simple one-to-one mappings between threats or security goals on the one hand and design choices on the other are rare. While implications may exist so that necessary security functions can be identified, it is often insufficient to implement only these functions—security is not only a set of features.

Security requirements comprise functional and non-functional aspects. To create a secure product, developers have to devise a security architecture, integrate a suitable set of security functions, and avoid defects that could be exploited to subvert the security architecture. Since such defects can appear almost anywhere, security is also not incremental. Further development can easily break a formerly secure product.

Many specific security requirements have, therefore, a technical rationale and only an indirect relationship to threats or security goals—they do not follow straightforwardly from a threat or goal statement. Such requirements are likely discovered by developers or security experts rather than by stakeholders, which has been observed more generally for non-functional requirements [5].

While security is a cross-cutting concern, the set of security aspects to be considered is non-uniform across development tasks. This can make it difficult to use checklists and catalogs like the Common Weakness Enumeration

(CWE, http://cwe.mitre.org/) or the OWASP Application Security Verification Standard (ASVS) [31]. Only a subset applies to any particular task and this subset keeps changing. Furthermore, detailed information about everything that may go wrong is not necessarily what a developer needs to prevent these problems.

Finally, security requirements engineering is subject to an epistemic challenge: beyond a baseline of common problems it is difficult, if not impossible, to know for certain how threats will react to the presence of a system. Security work aims to prevent future damages [32], but feedback from the field comes necessarily late. The benefit of security can therefore be difficult to asses and even for security functions that are clearly beneficial, customers' willingness to pay varies [33].

3.2 Security Development Work

Security is a secondary aspect to every part of the development cycle. In addition to the development tasks aiming to fulfill functional and other requirements, development teams have to make deliberate efforts to control the security properties of their product:

Requirements: As a basis for design decisions, developers need to understand the operational environment of a system, pertinent threats and risks, the stakeholders' security goals, existing security policies, etc. The analysis of such information leads to assumptions to rely on and to priorities for further security engineering. Some security functions may already be requested at this stage.

Architecture: Developers need to devise a viable security architecture, combining security mechanisms such that they build on each other and are not easily subverted. The security architecture defines aspects like trust relationships and the attack surface of a system.

Design: Security design concerns aspects like the selection of particular variants of a general type of security function (e.g., a specific user authentication scheme), the design of security-related user interfaces, and the design of internals like programming interfaces or the location within the architecture where certain functions are to be implemented. Tradeoffs can arise between different security needs or between security and other requirements, which must be resolved as part of the design process.

Implementation: As otherwise minor defects can easily break a security architecture, implementation work must aim to prevent such defects. They can arise in the implementation or application of security functions, but also in the implementation of any other part of a system if the security architecture explicitly or implicitly relies on it.

Testing and verification: To uncover vulnerabilities, specific testing and evaluation techniques are needed. Some types of implementation defects can be found with automated security scanners. Deeper issues may be found by penetration testing [34] or by code and design reviews, which require more

effort. Found vulnerabilities should be analyzed to identify and fix their root causes.

Security needs continuous attention throughout development. However, the effort that can be spent on security as a secondary objective is necessarily limited.

4 Scrum

4.1 A Management Framework for Software Development

Scrum [6,7] specifies a framework for agile software development. A Scrum team comprises a cross-functional development team, a product owner, and a scrum master. The development team is responsible for all technical aspects and development work, the product owner steers the direction of the project through ranked requirements, and the scrum master takes care of the process and facilitates interaction. Development proceeds as a series of iterations (sprints) and team members coordinate their work in ceremonies that repeat every sprint or even daily. At its heart, Scrum is a management framework for development work rather than a development process. Its distinctive elements are not short iterations and daily standup meetings, but rather:

Division of roles and responsibility: Scrum assigns precise roles and responsibilities to the development team, the product owner, and the scrum master, gives each role autonomy to carry out their respective task, and organizes interactions between actors in specific ceremonies. This precludes micromanagement of development work while allowing the direction of the project to be managed. The product owner conveys *what* shall be worked on in which order, whereas the development team has authority over the *how* of development. Coordination takes place in sprint planning and reviews. Development work is defined, prioritized, and executed in a loop between the product owner and the development team.

Handling of functional and quality requirements: In the logic of Scrum, functional requirements belong into the product backlog and their importance is defined by the product owner. Quality requirements belong into the definition of "done" and their fulfillment is to be demonstrated during sprint reviews. Scrum does not specify how to deal with requirements that do not fit into this clear-cut separation or are difficult to demonstrate.

Value-oriented backlog ordering: Scrum requires the product owner to order the items in the product backlog by expected business value. In line with agile principles, more valuable work thus gets precedence over less valuable work. This assumes, however, that the relative value of each backlog item can be determined at all and be expressed on the same scale.

Scrum defines only essential roles, artifacts, and interactions, so that by Scrum alone the work environment is underspecified. For example, the product owner is a generic proxy for project stakeholders. Whether the product owner

represents a customer, the product management in a company, or diverse stakeholders is not defined by Scrum. Likewise, work practices and routines are only specified to the extent necessary for Scrum as a management framework. Any number of further agile practices may be used in conjunction with Scrum, and development teams likely follow routines that either refine those defined by Scrum or run in parallel. For example, Scrum teams may handle defect reports and feature requests in different ways [15]. Moreover, further organizational structures surround Scrum teams in companies. For example, larger software companies often employ product security teams to support developers. Scrum constrains the interference of such structures with development.

4.2 Case Study: Security Work in an Agile Product Group

To research the emergence of security work practices in an agile development group we carried out in previous work [15] a field study at a large global software vendor. The investigated product group with five Scrum teams was embedded in an organizational structure with a dedicated product management and an R&D management. Both collaborated to set up the product strategy, to negotiate on the priorities of feature development, and to coordinate the development process together with the Scrum teams.

We followed the product group for 13 months with questionnaires, observations, document analysis and interviews, while the developed product underwent a security penetration test followed by a security training workshop for the developers. Our study explored the research question how the organizational routines of the product group changed to account for additional security work practices during and after the penetration test and training. We found that both events had a visible impact on the product group members, who became security aware and were determined to solve the vulnerabilities the penetration test revealed. However, despite group members' evident belief that security work should be part of development practices, they did not manage to change their organizational routines to account for it. Security work remained a reactive and unguided activity executed as unsystematic, invisible work task for individual developers. Developers themselves questioned this outcome and challenged whether security activities would actually be performed in the long run.

We concluded that in order to succeed, approaches to encourage and anchor security work in a development setting must be aligned to the setting's defining organizational aspects. This requires to not only consider a development framework such as Scrum as a mere constraint for secure software development interventions, but to also address its implications on how work is structured, understood and performed by developers and other people like R&D and product managers, as well as the embedding of development work in a broader organizational context. In particular, we found that as security requirements emerge from activities during product feature development, it might be considered a quality aspect of the software, but, however, cannot be handled as such as it actually requires continuous visibility and prioritization by other stakeholders outside the development team such as the product management. These key observations

motivated us to reconsider current proposals for secure software development in agile development environments, and particularly for scrum.

4.3 Consequences for Security Work

The way Scrum organizes software development has several implications for security work, which we observed in practice in our previously mentioned case study [15]:

Security must be visible and tangible. Scrum uses two key artifacts to manage development: the product backlog for requirements and tasks, and the definition of "done" for cross-cutting qualities. These artifacts represent what is deemed important in a project. If security does not appear there, or only in an ineffective way (e.g., as backlog items that never make it into sprint backlogs or as a security definition of "done" that is not actionable), substantial and systematic security work is unlikely to take place.

The product owner plays a key role. The product owner steers the work and the attention of the development team. Whether visible security concerns will be addressed depends on the ranking of security items in the product backlog and on the acceptance criteria effective in sprint reviews.

Requirements and priorities come from stakeholders. Stakeholders like clients or product management are the ultimate source of requirements and priorities. Security concerns must therefore be *their* concerns to be addressed in development. Since agile development explores requirements interactively with the stakeholders, they must must be able to assess the fulfillment of their requirements, to give feedback, and to express their valuation of requirements. As agile development avoids speculative and preemptive work beyond the scope of the current cycle, only continual feedback can keep concerns and requirements alive.

Security practices need to emerge, not be prescribed. Prescribing security practices to be followed by a team or individual developers, collides with two elements of Scrum. Such prescriptions would circumvent the product owner and the process to interfere directly with the work of the development team, which is self-organizing and uses retrospectives to reflect and improve its practices. Hence, prescribing practices from the top down introduces ambiguity into the self-management duties of the teams and the responsibilities for work tasks and results.

5 Managing Security Work in Scrum

Scrum supports three different ways to handle security concerns: security as bug fixing, security as a quality requirement and security as a set of features. In the following, we explore for each approach its implications of applying it in the context of agile development with Scrum, using our field experience (cp. 4.2) and other prior work. We show tensions and pitfalls we think require our attention as researchers with an interest in software development.

5.1 Security as On-Demand Bug Fixing

Like other defects, vulnerabilities may be discovered outside the immediate development process and reported to the team. Common sources of such reports include operators of the product, penetration tests and security reviews commissioned by vendors or customers, bug bounty programs, and incidents observed in the field. Such reports often concern specific instances of vulnerabilities, which may be isolated defects or consequences of deeper problems that likely cause further instances of similar vulnerabilities.

Bug reports arrive asynchronously as defects are being discovered. Bug fixing is a common activity in software development. Scrum treats bug reports as backlog items like any other. However, teams and organizations may develop practices to address urgent bugs quickly, without having to wait for the next sprint for proper planning.

Fixing known vulnerabilities is a necessary part of security development, but by itself insufficient to raise the security of a software product. Once an exploitable vulnerability has been discovered, it must be patched to prevent foreseeable or even to stop ongoing attacks. However, if this only forces attackers to look for another instance, security does not really improve. Bug fixing likely misses root causes that lead to numerous similar vulnerabilities. Patterns remain invisible when individual defects are treated in isolation and mitigating root causes often requires changes beyond the scope of a bug fix.

5.2 Security as a Quality Requirement

Software security has a strong quality aspect [21,3]. The nominal presence of required security functions does little to secure a system if they are not effective, and the effectiveness of any security function is easily undermined by software defects in the function itself or elsewhere in the system. Vulnerabilities in a security function weaken it and vulnerabilities elsewhere may allow its circumvention. Security therefore has to be considered as a cross-cutting, non-functional quality requirement.

Scrum's mechanism for quality assurance comprises the definition of "done" together with sprint reviews. The definition of "done" specifies a quality gate that increments have to pass before being accepted as completed. A typical definition of "done" specifies several tasks that must be accomplished in addition to implementation, such as documentation, code review, testing, and fixing of any known defects. These tasks are part of the development team's responsibility. Their completion results must be demonstrated or at least declared during the sprint review for the product owner to accept a backlog item. Definitions are adapted to the needs of a team and its product, but not to individual backlog items.

To address security as a quality requirement within Scrum, one would have to include generic yet effective security verification tasks in the definition of "done." Developers would write secure code to the best of their ability and verify their results by suitable means. Alas, this approach has limitations:

Developers cannot cope. There is a vast number of potential security concerns, of which only a subset applies to any particular development task, and this subset varies across tasks. It is unrealistic to expect developers to "just care" about security, to mentally select and apply the right set of secure development guidelines in everything they do. Security requires explicit efforts rather than just implicit attention.

Security is not only a quality attribute. Security comprises a mix of quality attributes, security functions, and security architecture. The functional and architectural aspects are not sufficiently covered by a quality-focused approach.

Direct verification is hard. Due to the low visibility of security and vulnerabilities, it is difficult to verify results. While a new function can be demonstrated and evaluated in a sprint review meeting, security properties evade quick and easy evaluation. Moreover, stakeholders and product owners are likely unable to provide useful feedback where deep technical expertise may be needed to even follow the discussion.

Testing has limitations. Indirect verification by demonstrating test or review results hinges on testing capabilities. Automated security tests, which fit into agile practices, work well for some types of vulnerabilities but miss others altogether. Penetration testing and thorough reviews, on the other hand, are costly and hard to repeat at sprint pace. When carried out asynchronously rather than within an iteration, their findings are likely handled through bug fixing.

While the development team can apply security practices and even adopt them autonomously as part of its self-organization, there is limited potential for feedback that could drive the team to do so. It may be tempting to mandate the creation of certain artifacts, e.g., "a threat model must exist for every new feature," but this would create new software bureaucracy, which agile development avoids.

5.3 Security as a Set of Features

Substantial amounts of security work are to be defined and scheduled through the product backlog and Scrum process just like any other kind of requirements. This kind of development work includes, for example, the implementation or strengthening of security functions, the design and implementation of general solutions to classes of vulnerabilities (see [35] for an example), and architectural improvements to reduce the potential for vulnerabilities.

For security feature work to be eventually done, it needs to be specified in the product backlog as a requirement or design idea, prioritized sufficiently as a backlog item to be worked on, refined until ready for implementation, and moved to a sprint backlog for execution. The implementation is then verified like for any other backlog item and the produced increment may be released to users and stakeholders, who can give feedback and raise new or updated requirements.

Perhaps with the exception of implementation itself, each of these steps can fail due to obstacles inherent to security:

Developer-defined requirements: For security requirements or ideas to appear in the product backlog at all, someone has to identify and specify them. This is unproblematic for common security functions with a clear and obvious business value, which may be requested by stakeholders just like any other kind of feature. However, these stakeholders are unlikely to specify less obvious or less common security functions, particularly when the need for these functions has a technical motivation that only developers understand. Scrum does not prevent developers from adding items they deem necessary to the backlog, but also does not specifically invite them to raise requirements for their product.

Prioritization of security work: Once listed on the product backlog, a security requirement will get further attention only if the product owner gives it sufficient priority. All backlog items compete with each other for limited attention and resources; only those with the highest business value have a chance of being implemented. Security work has several disadvantages here. First, its value proposition differs from that of features with positive business value. Security work limits losses at some later time [36,32]. Second, the difference between secure and insecure is hardly visible, so that even an unequivocal value can be difficult to argue. Third, proposals coming from developers likely lack a stakeholder who would champion them. These factors together make it less likely for security work to receive high priority.

Necessary expertise: Refinement, design, and implementation of security features can be inhibited by limited security expertise in the development team. Scrum assumes cross-functional teams, which have all required skill and expertise on the team to accomplish their development work. However, security has manifold, diverse facets and it can be difficult to cover all of them. Simultaneously, detail often matters in security and any approximation of a good solution can be as insecure as no solution at all.

Features alone are insufficient: Verification has two levels. Functional verification, checking whether a function has been implemented as specified, works as it does for any other function. However, correctness does not imply security, so that the effectiveness of a security feature needs to be verified in addition. This entails the challenges of security as a quality requirement discussed above.

In addition to these obstacles for security within the management process of Scrum, obtaining stakeholder feedback on security is difficult as well, because security properties and vulnerabilities are not readily visible.

6 Reconciling Scrum and Security

For all we know, Scrum works well as a development management framework. As development progresses and security needs evolve, the dynamic, empirical and change-accepting approach of agile development to requirements engineering should be suitable for security development. However, security as a class of

requirements has special properties that do not align with Scrum's strengths. Three different ways of identifying and addressing security concerns would be compatible with Scrum as discussed above, but each has limitations. Mere bug fixing is widely rejected as an approach to secure software development. Handling security as a quality requirement suffers from a lack of feedback within the Scrum process and neglects the functional and architectural aspects of security. Security work can be scheduled and managed like other backlog items, but does not fit into the scheme of stakeholder requirements prioritized by business value. Software development projects may therefore fail to address relevant security concerns contrary to the objective interests of their stakeholders. Our analysis suggests that remedy may be found in the following areas:

Reflective discovery of security needs: Test results and vulnerability reports can reveal unfulfilled security requirements. To identify and address these requirements, it is necessary to review collections of defects, look for patterns and find common causes. Such a review differs in scope and purpose from defect fixing itself; it aims to create new product backlog items that, if implemented, reduce the risk of future occurrences of similar vulnerabilities. Activities and tools that let teams reflect on security could be beneficial and would also fit into the general framework of agile development.

Value and prioritization of security work: Scrum prioritizes work by its expected business value, but security work does not fit into this approach. This creates a barrier, which systematically disadvantages some aspects of security development work. Security would benefit from better means for developers, product owners, and stakeholders to make the benefits of security work items visible at all and comparable with the value of other backlog items.

Security expertise and security consultancy: Thus far there seems to be little work on security expertise in Scrum teams, its impact on development results, and suitable ways of dealing with the diversity of security concerns. However, many software companies have established separate security teams as well as training programs for their developers. This calls for empirical work taking a closer look at interaction, collaboration, and impact.

Verification and feedback: Finally, verification and feedback for security need more attention. Agile development depends on short feedback loops, but quickly generating useful security feedback remains a challenge. Users and other stakeholders cannot provide feedback on security properties. Automated tests are quick and easily repeated, but for many security concerns not reliable enough. Penetration testing can deliver high-quality feedback, but requires too much effort to repeat frequently. Better feedback mechanisms would help Scrum teams to notice and address security needs.

These four proposals are neither magic bullets nor should they replace any of the security practices that can be applied within agile development. Their aim is, rather, to amend a set of software engineering activities with suitable agile management practices for the security aspects of software development.

7 Conclusion

Scrum as a management framework for software development establishes a control loop between the product owner and the development team to define, prioritize, plan, and review development work. We analyzed how software security fits into this framework and found several factors that can impede security work. Conceived as a quality attribute, security is too complicated and not visible enough to manage it effectively through the definition of "done" and sprint reviews. As a set of backlog items, security work needs to be defined by the development team but also prioritized by the product owner. We expect security development with Scrum to benefit from reflective practices to discover security concerns, better ways to value and prioritize security work, empirical results on internal and external security expertise, and better verification and feedback tools. Each of these four items deserves further exploration.

References

1. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for agile software development (2001)
2. Beznosov, K., Kruchten, P.: Towards agile security assurance. In: Proc. New Security Paradigms Workshop. NSPW '04, New York, NY, USA, ACM (2004) 47–54
3. Goertzel, K.M., Winograd, T., McKinley, H.L., Oh, L.J., Colon, M., McGibbon, T., Fedchak, E., Vienneau, R.: Software security assurance: A state-of-the-art report. Technical report, IATAC & DACS (July 2007)
4. Ramesh, B., Cao, L., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal **20**(5) (2010) 449–480
5. Ameller, D., Ayala, C., Cabot, J., Franch, X.: How do software architects consider non-functional requirements: An exploratory study. In: Requirements Engineering Conference (RE), 2012 20th IEEE International. (2012) 41–50
6. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Agile Software Development. Prentice Hall (2002)
7. Schwaber, K., Sutherland, J.: The Scrum guide. `http://scrumguides.org/` (July 2013)
8. Williams, L.: What agile teams think of agile principles. Commun. ACM **55**(4) (April 2012) 71–76
9. Boström, G., Wäyrynen, J., Bodén, M., Beznosov, K., Kruchten, P.: Extending xp practices to support security requirements engineering. In: Proceedings of the 2006 international workshop on Software engineering for secure systems. SESS '06, New York, NY, USA, ACM (2006) 11–18
10. Williams, L., Meneely, A., Shipley, G.: Protection poker: The new software security "game";. Security Privacy, IEEE **8**(3) (2010) 14–20
11. Ben Othmane, L., Angin, P., Weffers, H., Bhargava, B.: Extending the agile development process to develop acceptably secure software. IEEE Transactions on Dependable and Secure Computing **11**(6) (Nov 2014) 497–509

12. Pohl, C., Hof, H.: Secure Scrum: development of secure software with Scrum. CoRR (2015)
13. Baca, D., Boldt, M., Carlsson, B., Jacobsson, A.: A novel security-enhanced agile software development process applied in an industrial setting. In: 2015 10th International Conference on Availability, Reliability and Security. (Aug 2015) 11–19
14. Türpe, S.: The trouble with security requirements. In: 25th IEEE International Requirements Engineering Conference, IEEE (2017)
15. Poller, A., Kocksch, L., Türpe, S., Epp, F.A., Kinder-Kurlanda, K.: Can security become a routine? a study of organizational change in an agile software development group. In: Proc. CSCW'17, ACM (2017)
16. Highsmith, J., Cockburn, A.: Agile software development: the business of innovation. Computer **34**(9) (Sep 2001) 120–127
17. Cockburn, A., Highsmith, J.: Agile software development, the people factor. Computer **34**(11) (Nov 2001) 131–133
18. Williams, L., Brown, G., Meltzer, A., Nagappan, N.: Scrum + engineering practices: Experiences of three Microsoft teams. In: Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on. (2011) 463–471
19. Hoda, R., Noble, J., Marshall, S.: Balancing acts: Walking the agile tightrope. In: Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering. CHASE '10, New York, NY, USA, ACM (2010) 5–12
20. Hoda, R., Babb, J., Nørbjerg, J.: Toward learning teams. IEEE Software **30**(4) (July 2013) 95–98
21. Chung, L.: Dealing with security requirements during the development of information systems. In: Proc. Adv. Inform. Syst. Eng. (CAiSE'93). Springer, Berlin, Heidelberg (1993) 234–251
22. Rindell, K., Hyrynsalmi, S., Leppänen, V.: Busting a myth: Review of agile security engineering methods. In: Proc. ARES'17, IEEE Comput. Soc (2017)
23. Kongsli, V.: Towards agile security in web applications. In: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications. OOPSLA '06, New York, NY, USA, ACM (2006) 805–808
24. Mead, N.R., Viswanathan, V., Padmanabhan, D.: Incorporating security requirements engineering into the dynamic systems development method. In: 2008 32nd Annual IEEE International Computer Software and Applications Conference. (July 2008) 949–954
25. Maria, R.E., Rodrigues, Jr, L.A., Pinto, N.A.: ScrumS: a model for safe agile development. In: Proceedings of the 7th International Conference on Management of Computational and Collective intElligence in Digital EcoSystems. MEDES '15, New York, NY, USA, ACM (2015) 43–47
26. Baca, D., Carlsson, B.: Agile development with security engineering activities. In: Proceedings of the 2011 International Conference on Software and Systems Process. ICSSP '11, New York, NY, USA, ACM (2011) 149–158
27. Sonia, Singhal, A., Banati, H.: FISA-XP: an agile-based integration of security activities with extreme programming. SIGSOFT Softw. Eng. Notes **39**(3) (June 2014) 1–14
28. Rindell, K., Hyrynsalmi, S., Leppänen, V.: A comparison of security assurance support of agile software development methods. In: Proceedings of the 16th International Conference on Computer Systems and Technologies. CompSysTech '15, New York, NY, USA, ACM (2015) 61–68

29. Bartsch, S.: Practitioners' perspectives on security in agile development. In: 2011 Sixth International Conference on Availability, Reliability and Security. (Aug 2011) 479–484
30. Thompson, H.H.: Why security testing is hard. IEEE Security and Privacy **1**(4) (july-aug. 2003) 83 – 86
31. OWASP: Application Security Verification Standard. v3.0.1 edn. (July 2016)
32. Xiao, S., Witschey, J., Murphy-Hill, E.: Social influences on secure development tool adoption: Why security tools spread. In: Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing. CSCW '14, New York, NY, USA, ACM (2014) 1095–1106
33. Sonnenschein, R., Loske, A., Buxmann, P.: Which IT security investments will pay off for suppliers? using the kano model to determine customers' willingness to pay. In: 2016 49th Hawaii International Conference on System Sciences (HICSS). (Jan 2016) 5672–5681
34. Palmer, C.C.: Ethical hacking. IBM Systems Journal **40**(3) (2001) 769–780
35. Kern, C.: Securing the tangled web. Commun. ACM **57**(9) (September 2014) 38–47
36. Peeters, J., Dyson, P.: Cost-effective security. IEEE Security & Privacy Magazine **5**(3) (May 2007) 85–87