Guidelines for Agile Security Requirements Engineering

Christopher Wood
Department of Software Engineering
One Lomb Memorial Drive
Rochester, NY 14623
caw4567@rit.edu

Gregory Knox
Department of Software Engineering
One Lomb Memorial Drive
Rochester, NY 14623
gmk8349@rit.edu

ABSTRACT

Modern software systems are at the level of maturity where a single patch or "quick fix" near the end of the development cycle will not provide a suitable alternative to including security into the entire development process. In fact, software security is often viewed as a quality metric, which therefore implies it is most appropriate to start thinking about it during the requirements stage of a project. This is especially true for projects that use agile processes to iteratively release incremental software versions to generate a quick return on investment. Therefore, it is important that such processes are accompanied with the most appropriate requirements engineering activities that enable them to effectively elicit, analyze, model, validate, and manage requirements through the iterations of the project lifecycle.

Fortunately, many security-oriented process methodologies that support the integration of security requirements into traditional development processes have been proposed in recent years. Among these methodologies is the Microsoft Security Development Lifecycle (SDL). However, most of these processes are based on the waterfall approach to development and don't easily merge with agile processes. Therefore, in this paper we study and report on the two paradigms of agile requirements engineering and security-oriented development processes, and then propose some recommendations for merging these two entities together to yield mutually beneficial outcomes for high measures of agility and security awareness throughout a project.

General Terms

Software Security, Agile Processes, Requirements Engineering

1. INTRODUCTION

With the growing pervasiveness of cloud- and Internet-based computing services, software security is becoming an ever increasing issue that must be taken into consideration by all varieties of applications. Software systems built around the Internet are perhaps the most susceptible to state-of-the-art security threats, and since the computing paradigm is in the midst of a shift towards cloud and other related Internet-based services with applications such as online banking, e-commerce, and health record storage, it is important that such security threats are taken into account while these applications are in development.

Modern software systems are at the level of maturity where a single patch or "quick fix" near the end of the development cycle will not provide a suitable alternative to including security into the entire development process. In fact, software security is often viewed as a quality metric, which therefore implies it is most appropriate to start thinking about it during the requirements stage of a project. However, since there are a variety of development processes and methods that are utilized in different environments, it is important that the most suitable security requirement elicitation activities are used to optimize the time spent in the requirements stage of the project. Therefore, it is vital that software teams apply the appropriate security requirements engineering techniques based on their specific development process in order to make the best use of the time and efforts of the software team.

Traditional software processes are based on the Waterfall development lifecycle, which consists of requirements, design, implementation, test, and deployment stages, in which stage transitions only occur once a stage is deemed complete or rework needs to happen (in which case the stage will roll back to an previous one). However, most software projects that target cloud- and Internet-based services are often more agile because they seek to iteratively release incremental versions of their software to generate a quick return on investment. As such, agile processes will be the focus of this analysis, in which we will examine requirements engineering in the context of agile processes using standardized methodologies. In particular, we will focus on the Microsoft Security Development Lifecycle (SDL) and how it can be applied to agile processes to aid software teams in the requirements engineering effort of a project.

2. FUNDAMENTALS OF AGILE REQUIREMENTS ENGINEERING

Perhaps the most significant difference between agile and waterfall-based processes is that the entire lifecycle is broken up into smaller, variable-sized lifecycle increments that end in a new version of the software project being developed. These small increments make up iterations, or sprints, that typically last one to four weeks. Each iteration of the project involves the team going through a full development cycle, including requirements analysis, design, development, and testing [1]. By completing a cycle during every iteration the team is able to more quickly and effectively adapt to changes that are either introduced by technological problems, design issues, or changes from the customer. A large and important part of this cycle is the requirements analysis stage, which is distinct from traditional requirements engineering in several respects.

Firstly, active stakeholder participation is crucial to effective elicitation, analysis, modeling, and validation of requirements in agile processes. Including the stakeholders in the requirements engineering process enables the team become familiar with the terminology they use in describing their needs and also allows them to quickly adjust to changes or new requirements that arise during a sprint. Both of these results ultimately yield quicker

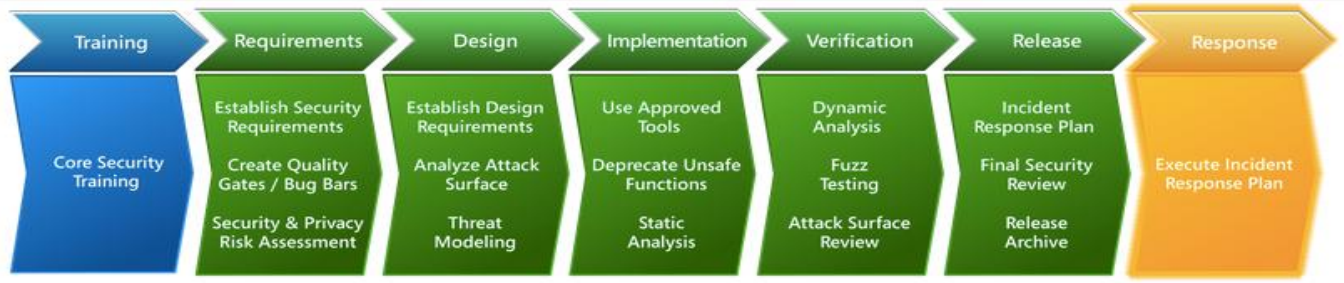

Figure 1: The Microsoft SDL lifecycle and its activities.

guarantees that the customer will be satisfied with the product they receive. Furthermore, active participation provides the team with the opportunity to jointly model system requirements by incorporating direct stakeholder feedback that emphasizes their specific needs the proposed system. This results in a more user-centered system specification and design, which increases the chances of stakeholder satisfaction.

Secondly, agile processes favor implementation over documentation, and requirements are no exception. Therefore, such processes do not have or promote thorough requirements documentation or specification. It is usually the case that too much documentation effort takes away from the team's ability to adapt, which may result in too much change overhead (or thrashing) across sprints. As a result, agile processes favor functioning specifications (often referred to as customer or development tests), which promote system functionality that have thorough amounts of implementation documentation. Such code-level documentation usually reflects the customer requirements to the same extent as formal requirements specification documents, but lack the change overhead that comes when the system functionality needs to change. Thus, the specification effort for requirements is migrated from static documents to functioning code that reflect the user needs and functional requirements.

Thirdly, the detail of requirements analysis that is performed in agile processes usually depends on the priority of such requirements. One conceptual similarity to this thought process is to treat requirements like a prioritized stack, in which higher priority items receive greater amounts of detail and lower priority items do not receive such detail. This thought process reflects both the Extreme Programming planning game and the Scrum methodology in that it involves the team creating a stack of prioritized and estimated requirements that need to be implemented. The team takes the highest priority requirements from the top of the stack that they believe can be implemented in the current iteration, freezes the requirements for the current iteration, and then treat changes as new requirements for the next iteration. This freeze on the requirements creates a stable set of requirements for the iteration to avoid confusion and inconsistencies [1].

Another difference in agile requirements engineering is that it is often seen as two stages; one activity that occurs at the beginning of a project and then smaller activities that occur at the beginning of every development sprint. The first requirements engineering stage is done following a breadth-first manner based on the priority of items being analyzed. The first requirements stage, which is often the largest and most extensive, aims at gathering requirements for the entire system so as to build and define a vision and scope for the product. Taking a narrow, depth-first approach in this first stage could make sprint development work difficult because the project increments would have been organized based on an incomplete vision for the project, which implies that any large-scale vision or scope changes could have drastic impacts on the sprint schedule and requirements stack.

Every-sprint requirements stages use the system vision and scope that was defined in the first stage to incrementally perform activities such as risk analysis and re-prioritization. The requirement analysis that takes place during each sprint allows the team to decide which requirements should be changed, removed, or added. It should not allow the team to go beyond the scope of the next iteration though; the goal is to keep the analysis short and on topic. By addressing the risks for each sprint the team is able to better adapt and manage any upcoming problems. As the project goes on and requirements are added, modified, or removed the risks can dramatically change. If not addressed these risks could get out of hand quickly, causing the project to lose momentum [2].

3. SECURITY ORIENTED PROCESSES

The Microsoft SDL was designed to aid software project teams in integrating security into their project and developing mechanisms at the policy, architecture, design, and source-code levels to realize the security requirements set in place by the project stakeholders. The principle idea behind this methodology is that governance, risk, and compliance activities are coupled among the activities in each stage of the lifecycle, which in turn means that software teams that follow this approach will usually satisfy their compliance requirements and manage risk effectively while also achieving high ROI in terms of security development [3].

The Microsoft SDL (which will be referred to as the SDL for simplicity) is a relatively new process that has been designed to be layered almost directly on top of traditional waterfall process stages, meaning that each stage in the SDL corresponds directly to a single stage in the waterfall model [4]. An overview of the stages and activities involved in each stage is shown in Figure 1.

From the process description there are three main activities involved in requirements engineering. That is, establishing security requirements, creating quality gates and bug bars, and performing security and privacy risk assessments. The first high-level activity is perhaps the most important step for developing and integrating security into any application. The security requirements must be identified, analyzed, formulated, and then validated against the stakeholder needs. The SDL expands this activity to include the assignment of security experts to various modules or aspects of the target system, definition of minimum security and privacy criteria for the application, and deployment of a security vulnerability tracking system that serves to enter, track, and watch the status of vulnerabilities as they arise during the development of the product or after it is released [4]. The vulnerability tracking mechanism is a very helpful tool that is often underused or replaced by common bug-tracking systems.

Establishing quality gates and bug bars is also an important step when eliciting requirements from stakeholders. In the context of security, certain software features may have very specific security requirements associated with them that must be satisfied before the feature is completed. For instance, most web applications require the user to log in to the system before use. While this is a very basic feature, it implies that the user must be authenticated using standard cryptographic techniques (such as password encryption and hashing) in order to make sure the private account credentials are not leaked to malicious users. Therefore, one quality gate for this feature would be that the underlying cryptographic techniques to support authentication are implemented correctly, tested, and validated.

In the same context, bug bars also aid software teams by throttling progression the development process lifecycle to match a certain amount of bugs (or security vulnerabilities) that have been identified. In other words, such bug or security vulnerability bars are set in place and if at any point in the development process these bars are exceeded, then development cannot continue until the levels are decreased to be below this threshold.

The last activity, which includes security and privacy risk assessments, is highly motivated by the similar need to assess traditional software system risks. Common software project risks are based in the context of the technology, software, hardware, schedule, cost, and people, as shown below in Figure 2 [5]. However, new software products require security to be an additional dimension in this paradigm, and by including this into the product lifecycle we must also include the appropriate risk assessment and mitigation techniques to manage it.

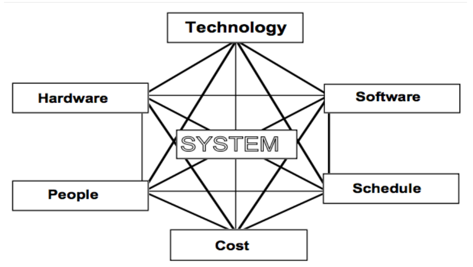

Figure 2: Six common dimensions of risk that are taken into account during the risk analysis activities of a project [5].

The SDL is not the only accepted process methodology that is accepted in the industry. In fact, the Comprehensive, Lightweight Application Security Process (CLASP), developed by the Open Web Application Security Project (OWASP), is yet another security-oriented process that defines additional roles and regulations on top of existing development processes. However, unlike the SDL, CLASP seems to place much more emphasis on the development portion of a project. While it emphasizes several useful requirements engineering activities, such as the application of misuse cases and identification of global security policies for all entities and resources in the proposed application, it is most concerned with maintaining high security metrics (i.e. keeping the number of vulnerabilities below a certain threshold) throughout the actual incremental development of the software. Thus, our focus tends towards the SDL because it seems to be more accommodating towards the actual security requirement elicitation and analysis stages of a project.

4. THE SDL AND AGLE PROCESSESES

Since the SDL is mainly designed for waterfall processes, it is not immediately applicable to agile projects. In light of this limitation, we present several requirements engineering activities that are motivated by the SDL that can be utilized in agile processes to incorporate security requirements into the project lifecycle.

4.1 Security Education and Awareness

Security awareness is critical for agile processes because it enables all parties involved to be capable of identifying, classifying, and responding to security issues as they arise during the elicitation (or even development) stages. Clearly, it is impossible to formulate any sensible list of security requirements if all parties involved don't understand the problem or aren't aware of any solutions. The National Institute of Standards and Technology (NIST) has well documented education curricula and activities that can be utilized by all stakeholders for a project prior to the requirements phase. Such learning programs focus on laws and regulations surrounding the use of the software, software security countermeasures (i.e. at the implementation level), as well as general security principles in the context of the entire lifecycle of a system

4.2 Emphasize Interviews and Product Research as Elicitation Techniques

In order to effectively identify security quality metrics that can be used to uphold the work on a software system, it is necessary to first elicit such security requirements from all stakeholders involved. Stakeholder elicitation techniques often consist of user surveys, job shadowing, brainstorming, product research, and interview sessions. However, for the purposes of security requirements engineering, interview sessions with the stakeholders and product research will yield the most return on investment in terms of preparation time and postmortem analysis. Interviews will give the requirements analysts the opportunity to gather desired quality metrics from the main business and user representatives, since these two stakeholder categories often have the most impact on the security of a product. Although security metrics defined as user needs are often more abstract and general than explicit functional (or nonfunctional) requirements, it places much needed emphasis on the elements of the system that are most important to the main stakeholders, and those that must absolutely be protected by the appropriate security mechanisms, as opposed mundane features of the system that don't warrant full-blown security requirements.

Product research is another important activity that is necessary if the SDL is to be applied to an agile process. If not to save time identifying common security threats and mitigation techniques that can be applied to their project, it helps provide the requirements analysis team with more insight into modern security threats that their product faces. This information, which may or may not extend beyond the educational training performed at the start of the project, can help the requirements analysts formulate proper questions for stakeholders pertaining to the security quality metrics that are sought after.

4.3 Extend Risk Analysis to Include Security

The risk analysis efforts for a project should be extended to incorporate security awareness from both the project developers and stakeholders. Identifying potential attack vectors, threats, and vulnerabilities that exist in a project design, implementation, or policy is absolutely necessary at every sprint, where such security threats are usually identified within the features of a system or along the boundaries of the system. This is extension is necessary because each sprint is scheduled to incorporate a small subset of functionality for the entire system, and with each additional feature comes the possibility for an exposed attack vector or vulnerability. Therefore, the risk analysis must incorporate such possibilities and the risk mitigation strategies must be validated at every stage to ensure that high agility is maintained in the event that such security risks arise.

Extending traditional risk analysis techniques to incorporate security does not change much of the process itself. Rather, it changes the depth of information that examined. A natural extension of the risk analysis techniques outlined in Section II is shown in Table 1 [6].

Table 1. Comparison of Risk Analysis for Security-Aware Agile Processes.

Traditional Agile Risk Analysis	**Security-Aware Agile Risk Analysis**
Identify	1. Identify the scope of the analysis 2. Identify and document potential threats and vulnerabilities
Classify	3. Gather data 4. Assess current security measures
Quantify	5. Determine the likelihood of threat occurrence 6. Determine the potential impact of threat occurrence 7. Determine the level of risk
Plan	8. Identify security measures and finalize documentation

Teams can also make use of the threat modeling tools and techniques provided by Microsoft in order to effectively identify risks that target their proposed application. It is important to note that, although threat modeling can be used as a catalyst to start the initial risk analysis effort in the requirements stage of a project, it must be done frequently with risk analysis throughout the duration of the entire project in order to be effective.

In addition, security policies can also be included in the requirements stage after risk analysis is completed. Such policies are usually a direct result of the threats and risks that face a system, so it makes sense to follow such risk analysis efforts up with identification of the security policies that will be implemented to avoid such threats and/or risks. These security policies can then be validated with the main stakeholders to ensure that they meet (i.e. do not violate) the user needs and functional requirements set in place during earlier requirements analysis stages.

4.4 Use Elected Security Officials or Standard Security-Vulnerability Tracking Tools

The best way to secure a software system is to assess and identify the vulnerabilities already in the system, evaluate the threats and then research ways to fix the vulnerabilities. Many organizations today employ multiple tools to identify their vulnerabilities from networks, applications, databases and source code, making them a staple for ongoing security management programs.

One of the tools that can help a company manage their security risk is Rsam Threat & Vulnerability Management Solution. Rsam enables organizations to record vulnerability data gleaned from existing scanning devices and automate remediation efforts, reducing the organization's risk and helping them stay compliant with regulations and policies. Rsam also enables organizations to centrally manage scanner results, providing security teams with a consolidated view of threat and vulnerability management initiatives across the organization.

Some key features of Rsam are:

- **Centrally Manage All Threat & Vulnerability Data** – Import data from one or many vulnerability scanning tools. Get a consolidated view of all vulnerabilities across the enterprise from a single console.
- **Risk-Based Workflows & Escalations** – Automate workflows based on existing, proven processes. Assign individual tasks and keep track of the progress throughout the remediation process. Send notifications and create escalation levels at appropriate areas.
- **Normalize Severity Ratings** – Translate risk ratings from multiple scanners into one rating scale for consistent reporting across your environment.
- **Prioritize Remediation Tracking** – Evaluate the risk of vulnerabilities based on other relevant risk data (asset criticality, compliance requirements, threats, etc.) to prioritize and determine appropriate action plans.
- **Dashboards & Report Trends** – Leverage out-of-the-box, role based views & dashboards; or create your own dashboards using a simple drag & drop interface.

Rsam would be useful for agile teams because it removes a lot of the overhead involved with tracking security risks, which in turn allows them to quickly adapt to project changes. Furthermore, it enables a single person or group of people to oversee the requirements tracking stage, so there is not shared thrashing among team members when trying to maintain the requirements.

4.5 Incorporate Misuse Cases

One dimension of common use case analysis models is the misuse of a system component or entity. Such misuse cases are directly applicable to both identifying and establishing security requirements within a project. The benefits of such analysis approaches are two-fold [7]:

1. Inform stakeholders about risks that target their application and communicate the rationale for proposed security requirements relating to certain resources or entities in the system.
2. Provide brainstorming and security requirement refinement opportunities for requirement analyst teams.

An intuitive description of a misuse case is that it is an inverted use case in that it is a function of the system that should not be allowed [7]. Using this same intuition, misuse cases are accompanied with both traditional actors and mis-actors, or those malicious users or entities that attempt to invoke system functionality that is not allowed. All known mis-actors should have very detailed and complete profiles so as to distinguish them from similar mis-actors in the context of a single misuse case. In addition, by the very nature of misuse cases, it is essential to support exceptional and alternate paths that can be taken by the system in response to actions taken by either legitimate or malicious actors.

Based on the need to track security requirements in the SDL, the following information should be accompanied with every exceptional and alternate path [7]:

1. Assumptions made during the flow of execution that led to this action being taken (i.e. system preconditions, user access rights).
2. Prevention and detection guarantees that will be made by the system in order to handle such paths.

One final extension to traditional use-case modeling is that the exceptional and alternate paths in misuse cases should be analyzed in an iterative (or recursive) pattern so as to ensure optimal branch coverage for all identified security threats that the system faces. Once these steps are done, it is again necessary to review the proposed analysis model with the stakeholders in order to determine any new security requirements that should be handled, policies that need to be put in place to enforce misuse detection and prevention guarantees, and validate existing security requirements that have already been identified.

4.6 Utilize Security Requirements Patterns

The concept of reuse is a familiar notion within the software development realm, but less common when considered in the context of requirements engineering. Requirements reuse from existing patterns provides organizations with the opportunity to share a requirement across projects without absorbing unnecessary duplication of artifacts within a repository. This is a critical capability that accelerates time to market and cuts development costs. Shared requirements can either track to the ongoing changes made by the author or they can remain static if the needs of the project dictate.

Security requirements patterns cover a broad spectrum of requirements engineering activities, and can extend from security policies to use-case definitions and descriptions (including misuse cases). However, given the rapid advancement of technology that modern software systems are based upon, these patterns sometimes only provide a general baselines for activities that require more application-specific details to be filled in from the requirements analysts and stakeholders. For example, one common security requirements pattern is the misuse case for a man-in-the-middle attack on a peer-to-peer architecture. While this pattern describes common actors in the use case, it does not usually include information relating to the communication channel that is used for peer exchanges, or the access rights and privileges that each user has. This is information that must be collectively filled in during the requirements stage in order to make the use case complete.

4.7 Consider the Entire Technology Stack

The application design and feature set of a software system have as much impact on the potential threats as the operating environment in which the system will run. The complexity of the software stacks that are used to construct modern software systems continues to grow at an alarming rate, and in order to perform thorough risk analysis of an entire system it is necessary to understand the attack vectors and vulnerabilities that exist in each layer of this stack.

Furthermore, having an understanding of the entire software stack will enable the team to formulate more realistic functional requirements and constraints put in place on the system. In turn, this will lead to less overhead during the requirements phase of each sprint as the amount of rework and change overhead should theoretically decrease. Unclear or ambiguous environment conditions can cause confusion among the team or inaccuracies in the functional requirements specification. Therefore, it is vital that these details are outlined in the first requirements engineering stage while the product vision and scope is being developed.

5. Conclusion

Agile process methodologies are very beneficial when applied to the right application, such as cloud- and Internet-based services. However, as the software world continues to evolve security requirements need to be more and more prevalent. It is important for teams to start including security requirements during the beginning of a project. By determining the security requirements early on in the project the team can avoid the typical “quick fix” that is used to patch problems that are found after the project has been completed. If a process like the Microsoft SDL is used for a project, the team will be better equipped to find, prevent, or fix any security flaws that are found in the project early on, which is the most cost effective time to fix all flaws.

6. REFERENCES

[1]- Lan Cao; Ramesh, B.; , "Agile Requirements Engineering Practices: An Empirical Study," *Software, IEEE* , vol.25, no.1, pp.60-67, Jan.-Feb. 2008. doi: 10.1109/MS.2008.1.

[2] - "Five Simple Steps to Agile Risk Management." *Michael Lant*. Web. 03 May 2012. http://michaellant.com/2010/06/04/five-simple-steps-to-agile-risk-management/

[3] - Howard, M.; , "Building more secure software with improved development processes," *Security & Privacy, IEEE* , vol.2, no.6, pp. 63- 65, Nov.-Dec. 2004 doi: 10.1109/MSP.2004.95

[4] - Sullivan, B. “Security Development Lifecycle for Agile Development”. Presented at *Blackhat Washington DC 2010 Conference* (BlackHat DC’2010). Washington, DC, USA. http://www.blackhat.com/presentations/bh-dc-10/Sullivan_Bryan/BlackHat-DC-2010-Sullivan-SDL-Agile-wp.pdf

[5] - Higuera, R. and Haimes, Y. "Software Risk Management." Software Engineering Institute - Carnegie Mellon University. Web. http://www.sei.cmu.edu/reports/96tr012.pdf

[6] - Department of Health and Human Services, *Basics of Risk Analysis and Risk Management*. Web. Mar. 2007. http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/riskassessment.pdf

[7] - Sindrew, G., Andreas, L. "Templates for Misuse Case Description." In Proceedings of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ’2001). Web. 03 May 2012. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.8190

[8] - SANS Institute, *Information security policy templates*. SANS Institute, n.d. Web. 3 May 2012. http://www.sans.org/security-resources/policies/

[9] - Gregoire, J., Buyens, K., De Win, B., Scandariato, R., Joosen, W. 2007. On the Secure Software Development Process:
CLASP and SDL Compared. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems* (SESS '07). IEEE Computer Society, Washington, DC, USA, 1-. DOI=10.1109/SESS.2007.7 http://dx.doi.org/10.1109/SESS.2007.7

[10] - Xiaocheng Ge, Richard F. Paige, Fiona A.C. Polack, Howard Chivers, and Phillip J. Brooke. 2006. Agile development of secure web applications. In *Proceedings of the 6th international conference on Web engineering* (ICWE '06). ACM, New York, NY, USA, 305-312. DOI=10.1145/1145581.1145641