Integrating Software Development Security Activities with Agile Methodologies

Hossein Keramati, Seyed-Hassan Mirian-Hosseinabadi
Sharif University of Technology
keramati_h@mehr.sharif.edu, hmirian@sina.sharif.edu

Abstract

Because of several vulnerabilities in software products and high amount of damage caused by them, software developers are enforced to produce more secure systems. Software grows up through its life cycle, so software development methodologies should pay special attention to security aspects of the product. This paper focuses on agile methodologies in order to equip them with security activities. We can restrain reduction of agile nature of organization's current process by means of agility measurement and applying an efficient activity integration algorithm with a tunable parameter named agility reduction tolerance (ART). Using this approach, method engineer of the project can enhance his agile software development process with security features to increase product's trustworthiness.

1. Introduction

Security is an essential quality aspect of software products and in some cases, it is the most important non-functional requirement of the system. Using software systems in critical environments and relying on its functionality may lead to huge amount of risks and damages if security weaknesses are present.

Although there are a lot of researches representing new methods, techniques, procedures, guidelines, and tools to increase security in software systems, an augmenting rate of security defects and vulnerabilities are reported every year anyway. Non-functional requirements including security cannot be injected in a system like functional features; the development team should pay special attention to security during all stages of software development life cycle. Analysis, design, implementation, test, deploy, production and also product's death iterations should be armed with security activities to reach more secure software.

In recent years, there are some efforts to a add security features to software development processes, evaluate them and also introducing modeling languages and tools helping project's team to produce secure systems. We briefed these works in the next section as related works.

In this paper, we present a method to add security activities to agile software development methodologies with a tunable parameter controlling agile characteristic of the process. Extracting security activities from existing resources, calculating agility degree of activities and the process, introducing agility reduction tolerance parameter as well as an algorithm to integrate security activities with activities of current agile process are main contributions of our work.

This paper continues as follow: next section shortly reviews related works on securing software development processes; agile methodologies are briefed in section 3 and then, our method is described in section 4. Some enhancements to this method for the future, conclusion and references form final sections of this paper.

2. Related Works

In response to increasing rate of damages to businesses caused by security vulnerabilities in software products, some efforts have been started to develop more secure software. These works led to some guidelines, best practices, design and modeling techniques, risk management methods and also a lot of tools for security analysis, test and so on. In addition there exist a few attempts on securing software development life cycle that are mentioned shortly in the following paragraphs.

AEGIS introduced by Flechais [5] is a methodology for developing secure systems. It has been designed based on asset modeling, security requirements identification, risk analysis, and usability context. AEGIS starts after design phase of software development life cycle and tries to reduce vulnerabilities by means of risk analysis and mitigation, but it does not represent any clear procedure for the development team so as to perform these activities. It is not a full-lifecycle software development process.

Microsoft's experiences in security of software products are collected and integrated as a process named SDL which stands for Secure Development Life Cycle [1]. This process consists of 13 sequential stages that cover most parts of software development life cycle.

978-1-4244-1968-5/08/$25.00 ©2008 IEEE

Another process for developing more secure software is a work done in the CLASP project [9]. Providing a set of activity based process components to be integrated with other software development processes is the goal of this project. CLASP 2006 composed of 24 security activities that are organized in seven categories named best practices.

Microsoft SDL and CLASP are two major process level efforts for enhancing security of software products. They have some advantages and disadvantages that are analyzed and compared by Gregoire and his colleagues in [2]. Both have some valuable activities to increase level of security, however, there are many shortcomings in them. They are starting points of secure software development processes not the goal.

For agile methodologies, Microsoft presents some guidelines to use Microsoft SDL's practices with agile methods [1]. Beznosov classified security assurance methods and techniques regarding their conflict with agile development, and made suggestions to relax them [3]. His main focus is on XP as an agile methodology [4].

SSE-CMM framework [13], Common Criteria [12] and NIST's published document [7] represent methods to evaluate security processes, products and activities. UMLSec project and its related researches is Jürjens's work at Oxford University [11]. UMLSec is a UML based modeling language for describing security aspects of software through its development process. TSP-Secure [14] for security planning and increasing security as well as a rich set of collected resources by US-CERT as BSI project [10] and another document from NIST [6] are other major activities in this field.

3. Agile Methodologies

In addition to traditional and object oriented software development processes, a new family of light weight methods named agile methodologies comes into existence with some interesting features. The goal of this approach is high development speed with more customer satisfaction. XP, Scrum, FDD, dX and DSDM are examples of these agile methods.

In agile methodologies, project's team are relatively small with too customer interaction and on the basis of communications between team members instead of heavy modeling and documentation overhead. Agile methods recommend developing software through various small iterations to analyze, design, implement, test and also release the product. These processes are composed of simple informal activities with lowest level of modeling and documentation, flexible, iterative and high of speed execution. Agile activities require oral communications among team members and are open to changes in requirements by way of direct customer interactions. These are features that distinct this type of methodologies from other seminal and integrated full-lifecycle processes. In practice, each activity with similar features can be integrated easily with agile methods. We will use this theory to extend agile methodologies by means of external security activities.

4. Extending Agile Methodologies with Security Activities

As mentioned earlier, there are some guidelines, best practices, methods and other materials that can be used by project's team to produce secure software products. To arm agile methodologies with security features, it is acceptable to use these experienced and proposed activities for secure software development. On the other hand, integrating some heavy weight activities with agile processes may lead to a process that cannot be named agile and possibly will be unacceptable for project's team. In order to restrain reduction of agility nature, we propose a method explained below that consists of five parts. First security activities are extracted from existing processes and guidelines, and then agility degree of activities is defined to measure their nimbleness. Integration issues of agile and security activities are handled and an algorithm to integrate security activities with organization's agile process is introduced; finally agility reduction tolerance (ART) parameter and its optimum value are discussed.

4.1. Extracting Security Activities

With a survey on works done in this field, we achieve a rich list of activities that can be performed during a software development process to reach more secure systems. In our study on several academic and industrial resources, about 70 activities and sub-activities ranging from project initiation to production and product's death are acquired. We name these activities as *security activities* and use this list as a basis for next steps in our method.

Classifying security activities will help us to understand each activity and its execution context better, and to see related activities in the same appropriate category as well. Project inception, analysis, design, implementation, test, deploy, and death are major phases of generic software development life cycle (SDLC). Figure 1 represents security activities positioned in the analysis phase of SDLC.

4.2. Calculating Agility Degree of Security Activities

In respect to agile manifesto [8], agile methodologies have common features that distinct them from other types of software development processes, including traditional

and modern object oriented methodologies. To extend an agile methodology by integrating an external activity with it, we should be careful of decreasing methodology's agile nature. Adding security activities to an agile method may leads to a heavy process that will not be performed as expected.

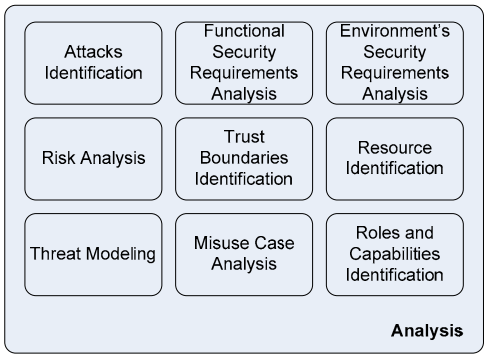

Figure 1 – Extracted Security Activities for Analysis Phase of Generic SDLC

By defining *agility degree* for each activity, we can measure its agile behavior. Agility degree represents level of activity's compatibility with agile methodologies and is calculated on the basis of agility features such as simplicity, rapidity, and people orientation. We consider major characteristics of agile methodologies and assign a grade between 0 and 5 to each of them, leading to a 1-culomn matrix named activity's *agility degree vector (ADVect)*. Great number for an agility feature of an activity means its high compatibility with that feature and a low grade shows a conflict. We have considered magnitude of following agility features as ADVect values:

$$ADVect \triangleq \begin{bmatrix} \text{Simplicity} \\ \text{Customer Interaction} \\ \text{Free of Modeling and Documentation} \\ \text{Change Tolerate} \\ \text{Speed of Execution} \\ \text{Informality} \\ \text{People Orientation} \\ \text{Iterative} \\ \text{Flexibility} \end{bmatrix}$$

For example, "Threat Modeling" is an activity in our security activities list, a complicated, semi-formal and heavy weight activity, which is based on modeling and requires in depth security knowledge. Detailed analysis of this activity results in estimation for ADVect of it as follow: [1 1 0 3 2 1 1 4 1]. It is an activity that is not as simple as agile methods, does not have great interactions with the customer, requires complete modeling and documentation efforts, with medium-level tolerance on changes, low speed, formal and method-based instead of people-oriented, and could be performed iteratively but inflexible.

Agility degree is sum of values in ADVect of an activity. This simple definition of agility degree gives a good sense about activity's respect to the agile manifesto and agile methodologies. Table 1 shows results of analyzing some security activities, their ADVect and agility degree as well.

Agility degree vector of an activity is not unique for all development teams and organizations. Execution of an activity may be difficult for a development team and simple for another. As a result, since there are differences among software development teams, their agile experience and their explanation of agility features, ADVect should be defined and calculated specially for each one. A committee named *secure method engineering team (SMET)* should perform this analysis and obtain ADVect for each security activity. SMET is composed of organization's method engineer who is responsible for SDLC methodology and software security engineer that is familiar with security activities.

4.3 Integration of Agile and Security Activities

Each agile methodology has its own core process engine that security activities should be run through this engine. To inject a security activity into an agile method, we should integrate this activity with one of the original method's activities. The first step is to analyze agile methodologies and identify their core engine activities.

As mentioned earlier, integration of external activities with an agile methodology and its activities may lead to a heavy non-agile new activity. Agility degree attribute has been defined to measure activities' agile nature. As well as security activities, we should calculate this value for extracted agile activities. We can compare agility features of these two types of activities and also estimate agility degree of a combined activity.

Integration of two activities with agility degree of *a* and *b* results an activity with *minimum(a,b)* as its degree. It is also true about ADVect of them. In addition, average agility degree of activities of a process is defined as *process's agility degree*.

It is impossible to integrate a security activity with all agile activities. Some activities could be combined to form a new activity, but it is not true about each pair of activities. For example "Vulnerability Test" is a security activity that its integration with "Final Test" activity of dX agile method, results a security armed test activity; hence, these two activities are integrable. As another example, vulnerability test cannot be combined with "Design" or even "Test-Driven Implementation" activities. We define another matrix named *activity integration compatibility matrix (AICM)*. It is a table with binary values in its cells; agile activities are on its row

Table 1 – Security activities and the grade of their agility features. Agility degree is calculated by summation of grades

Agility Feature / Security Activity	Simplicity	Customer Interaction	Free of Modeling and Documentation	Change Tolerate	Speed of Execution	Informality	People Orientation	Iterative	Flexiblity	Agility Degree
Attacks Identification	2	3	3	4	2	5	2	5	4	30
Threat Modeling	1	1	0	3	2	1	1	4	1	14
Security Requirements Analysis	4	4	3	3	4	5	4	4	4	35
Security Education & Awareness	3	4	5	5	5	5	5	5	5	42
Build Security Team	4	4	5	5	5	5	5	5	5	43
Resource Identification	2	3	3	1	3	3	2	4	3	24
Roles Identification	3	4	3	2	3	4	4	4	3	30
Review Design Security	3	3	2	3	5	4	4	5	5	34
Static Code Analysis	5	5	2	5	5	4	5	5	1	37
Penetration Testing	0	3	2	4	3	4	4	5	5	30
Incident Response Planning	2	5	3	3	2	4	4	2	4	29

and security activities form table's columns. For compatible activities, 1 will be inserted into their cross cell and 0 denotes two agile and security activities that are not integrable.

Agility reduction tolerance (ART) is a new parameter used in our algorithm which controls inserting of low agile security activities to an existing agile software development process. ART is a decimal number greater than 0 and lower than 5. A high value of ART shows that organization or project's team could accept heavy-weight security activities in their software development process so as to produce more secure software products. We will explain this parameter more and techniques to decide on it later.

4.4 The Activity-Process Integration Algorithm

Here is an algorithm to select security activities in order to be integrated with compatible current organization's agile process activities and improve security of software development life cycle. It consists of six steps and controls agility attribute of the new process by means of ART parameter as declared before.

Step 1: Select an activity from security activities list with highest agility degree. Assuming *SecActsList* and *ad(x)* as list of security activities and agility degree of activity *x*, we have:

$$secAct = x | \, x \in SecActsList \wedge \forall_{y \in SecActsList} \cdot ad(x) \geq ad(y)$$

Step 2: Using filled AICM matrix, form a list of agile activities that are integrable with selected security activity in step 1. Then from this list, select the heaviest one (lower agility degree) to integrate with:

$$aglAct = x | \, x \in AglActsList \wedge AICM[x, secAct] \neq 0 \wedge$$

$$\forall_{y \in AglActsList} \cdot ad(x) \leq ad(y)$$

where aglActsList is the list of activities extracted from current agile process. If there is no compatible agile activity to *secAct*, remove it from the list of security activities and go to step 6.

Step 3: Integrate selected agile and security activities and generate a new activity as *newAct* with new agility degree:

$$ADVect_{newAct} = min\big(\, ADVect_{secAct}, ADVect_{aglAct}\big)$$
$$\Rightarrow ad(newAct) = min\,(ad(secAct), ad(aglAct))$$

Step 4: Replace new activity with original one in the agile process and recalculate its agility degree. If original process's agility degree plus *ART* parameter is greater than or equal to new process's agility degree, this integration will be acceptable. Otherwise, selected security activity should not be inserted into current agile software development life cycle.

Step 5: Remove selected security activity (secAct) from the list of security activities (*SecActsList*).

Step 6: If there are some activities in the list of security activities, go to step 1. If not, algorithm terminates.

This algorithm leads to a new software development process for project's team that some security activities are integrated with it and agile nature of the method is controlled by means of ART parameter as well.

4.5 Agility Reduction Tolerance (ART)

Activities to increase security in software life cycle are usually along with high amount of effort. They may require documentation, formal and semi-formal operations and also use of some modeling procedures to foresight, prevent, detect and remove security defects from the software. This behavior varies from one security activity to another, but there is a form of heaviness in most of them. Therefore, blind integration of security activities with agile methodologies may lead to unacceptable decrement in agility of software development team, imposing too cost to the project and development speed reduction as well. On the other hand, security in produced software is an important issue and to achieve it, we should tolerate amounts of cost and agility level reduction; it is a tradeoff between security and the cost.

As mentioned earlier, agility reduction tolerance (ART) parameter defined as a criterion to control agility degree of existing agile process of the project. In the algorithm specified in previous section, security activities have an attribute named "agility degree". To integrate these activities with agile methodology, we start from the most agile activities to the heaviest one. Therefore we have minimum rate of reduction in agile nature of the process through procedure execution.

Tuning ART parameter is SMET's art to keep a balance between security and weight of the software development process. SMET should be able to measure cost caused by increasing the value of this parameter In one hand and amount of damages from security defects in the software product on the other hand. These damages can be eliminated by executing some security activities in development life cycle of the software. Using risk analysis methods to identify security risks, their potential damage and occurrence likelihood is a standard way to evaluate costs of vulnerable software. Also based on project's team and current agile process, method engineer can measure cost of agility reduction in his development process and finally with a cost/benefit analysis, calculate ideal value for ART parameter to inject some security activities in existing process.

5. Method Enhancements and Future Work

We can enhance the method introduced in this paper with some changes to it and new concepts. In calculating agility degree of the agile process, we can assign activities a weight based on their frequency in software development life cycle and have a more real agility degree of the process. Analyzing security activities and relax them to gain a better agility degree is another effort that SMET team may do. Making activities consistent, use of fuzzy values for AICM matrix and finally, dependency check among security activities are other enhancements that can be applied. This work is in progress.

6. Conclusion

Security is an important quality aspect of software systems and to achieve this goal, we should attend to it during software development life cycle. Using introduced method in this paper, security activities can be integrated to agile methodologies to enhance security of software product. Secure method engineering team (SMET) can tune agility reduction tolerance (ART) parameter to make a balance between costs of decreased level of agility because of security activities and benefits from developing more secure systems.

7. References

[1] Howard, M., Lipner, S., *"The Security Development Lifecycle - SDL: A Process for Developing Demonstrably More Secure Software"*, Microsoft Press, 2006.

[2] Gregoire, J., Buyens, K., Win, B. D., Scandarioto, R., Joosen, W., *"On the Secure Software Development Proecss: CLASP and SDL Compared"*, In Proceedings of the Third International Workshop on Software Engineering for Secure Systems, 2007.

[3] Beznosov, K., Kruchten, P., *"Towards Agile Security Assurance"* In Proceedings of the 2004 Workshop on New Security Paradigms, 2005.

[4] Beznosov, K., *"Extreme Security Engineering: On Employing XP Practices to Achieve 'Good Enough Security' without Defining It.",* First ACM Workshop on Business Driven Security Engineering, 2003.

[5] Flechais, I., Sasse, M. A., Hailes, S. M. V., *"A process for developing secure and usable systems"*, In Proceedings of the 2003 Workshop on New Security Paradigms, 2003.

[6] Grance, T., Hash, J., Stevens, M., *"Security Considerations in the Information System Development Life Cycle"*, NIST, Computer Security Division, NIST Special Publication 800-64, REV. 1, 2004.

[7] Swanson, M., etc, *"Security Metrics Guide for Information Technology Systems"*, NIST, Computer Security Division, NIST Special Publication 800-55, 2003.

[8] Agile Alliance, *"Manifesto for Agile Software Development"*, 2005, http://www.agilealliance.org.

[9] Secure Software Inc., *"CLASP: Comprehensive Lightweight Application Security Process"*, Version 2.0, 2006, http://www.securesoftware.com/process.

[10] US-CERT, Software Engineering Institute, *"Build Security In"*, 2006, https://buildsecurityin.us-cert.gov.

[11] Jürjens, J., *"Developing Secure Systems with UMLsec From Business Processes to Implementation"*, UMLsec homepage, 2002, http://www4.in.tum.de/~umlsec.

[12] The Common Criteria Portal, 2007, http://www.commoncriteriaportal.org.

[13] Systems Security Engineering – Capability Maturity Model (SSE-CMM) official web site, 2007, http://www.sse-cmm.org.

[14] TSP for Secure Systems Development – Presentation, 2002, http://www.sei.cmu.edu/tsp/tsp-secure-presentation.