Integrating Security into Agile Development Methods

Mikko Siponen[a], Richard Baskerville[b] and Tapio Kuivalainen[a]
[a]*University of Oulu, Department of Information Processing Science, Linnanmaa, PO BOX 3000, FIN-90014 Oulun yliopisto, FINLAND.*
[b]*CIS Department, Georgia State University, 35 Broad Street, PO Box 4015, Atlanta, GA 30302*

Abstract

Software developers can use agile software development methods to build secure information systems. Current agile methods have few (if any) explicit security features. While several discrete security methods (such as checklists and management standards) can supplement agile methods, few of these integrate seamlessly into other software development methods. Because of the severe constraints imposed by agile methods, these discrete security techniques integrate very poorly into agile approaches. This article demonstrates how the security features can be integrated into agile methods.

1. INTRODUCTION

The software industry has formalized agile product development [24] in various forms of agile software development methods [1]. For competitive reasons, developers often use these methods for web and network applications where security risks are prominent. Despite the prominent risks, the existing agile methods have few features specifically addressing security risks. As a result, agile software products will lack security protection unless such protection is added afterwards. Subsequently adding of security features to software requires courage from software developers and administrators. One reason why the agile methods ignore security issues may stem from a misconception that it is, indeed, security that hinders the development [25]. This is true with most of the existing security methods [3, 4, 22]. Our research indicates that developers can integrate security seamlessly in agile software development. For this purpose, security techniques need to meet several requirements, including that the security techniques must be adaptable and agile in order to operate in unstable conditions [3, 5].

This article describes the requirements for security techniques to integrate seamlessly into agile methods. We demonstrate these requirements with an example of an approach for adding security into agile software development methods. In practice, this solution has been successfully tested in early design phases [22]. We see that this approach also offers a workable solution for adding security in agile software development. We show how this approach can be adapted to the phases of agile methods.

2. SECURITY REQUIREMENTS FOR AGILE DEVELOPMENT METHODS

Agile software methods are characterized by nimbleness to rapid changes, multiple incremental iterations and a fast development pace [1]. This raises four requirements for security methods that are targeted to be integrated into agile software methods:

1. Security approach must be adaptive to the agile software development methods;
2. They must be simple; they should not hinder to the development project;
3. Security approach, to be integrated successfully with agile development methods, should offer concrete guidance and tools at all phases of development, i.e., from requirements capture to testing.
4. A successful security component should be able to adapt rapidly to ever-changing requirements owing to a fast-paced business environment, including support for handling several incremental iterations.

Several alternative security techniques have been proposed for the purposes of adding security considerations to general software development methods [4, 11, 21]. These include logical modeling [4], a soft approach [12], a security spiral [9], responsibility modeling [2], information modeling [16, 17, 18], abuse cases [13], an information systems security planning methodology [23]. None of these previous approaches particularly address these agile requirements [5]. To address these issues, we propose a set of key security elements, derived from on our earlier work [22], which form a generic but modifiable security process, which can be added seamlessly to agile software development methods.

3. KEY SECURITY ELEMENTS IN AGILE SOFTWARE DEVELOPMENT

Table 1. Key security elements in agile software development.

Key security elements
Security-relevant subjects
Security-relevant objects
Security classification of objects and subjects
Risk management

The key security element stems from information security "meta-notation," or notation for notations, [22] and database security [20]. We use a risk management framework in modified form from Saltmarsh and Browne [19]. We apply these key security elements to a process aimed at developing secure software in an agile manner. This generic security process consists of these key security elements in different phases of software development (requirements analysis, design, implementation and testing). These steps are not necessarily sequential and in any case, every step is optional.

Requirements analysis phase

1. *Identify security-relevant objects from requirements analysis*.

In the requirements analysis phase developers identify the key assets of the organizations. Discussions with customers reveal the security sensitive assets of the organization and enables developers to list these assets. These assets are candidate security objects, and are denoted in the notation of the agile method, e.g., security enriched use cases [22]. This notation documents the security-relevant objects.

2. *Identify security-relevant subjects from requirements analysis*.

In the requirements analysis phase there is a need to identify the authorized persons or groups who may accidentally or intentionally compromise the software security. Further customer discussions reveal the authorized persons in the organization. These persons are candidate security subjects. Again, these are denoted in the notation of the agile method such as the security enriched use cases. This notation consequently documents the basis for determining the security-relevant objects and the actors who are the potential security subjects.

3. *Determine the security classification of these security-relevant subjects and objects*.

Developers also need to determine the most crucial or sensitive security objects. They classify these security objects so that the most sensitive security objects (i.e., the assets of the organization) are labeled, e.g., "top secret" or "confidential" or "unclassified". Developers then analyze which security subjects, i.e., authorized persons, should have access to which security objects (i.e., assets of the organization). In our example, use case notation is also useful for this case. By constructing candidate use cases at the same time as checking security classification, developers ensure that the classification of the security objects and security subjects are correct.

4. *Do risk analysis and management*

Having identified the necessary security subjects (authorized people) and security objects (security sensitive assets of the organization), as well as the security classification of each according to its security sensitivity, developers must then analyze potential threats. For agile methods, we have found "abuse cases" effectively document potential threat scenarios [13, 14] and help identify non-permissible situations during software use. We also represent the estimated cost of recovery from an occurred attack to security objects (security-sensitive assets of the organization) to the abuse case scenario. This cost value enables a simple risk analysis and management process. The following risk analysis and management process summarized from Saltmarsh and Browne [19] to suit agile development:

i) What unwanted actions – risks – could happen to the assets of the organization i.e., what are the risks?
ii) If those risks, possibly listed in terms of abuse scenarios, would occur, what damage would they cause? Here the replacement costs of assets are calculated and added to the abuse cases.
iii) How often may the threat scenario be expected to happen? How reliable is this information?
iv) What can be done (countermeasures) in order to minimize the occurrence of these risks?
v) What costs would be incurred by these countermeasures? Is the implementation of these countermeasures economically and rationally feasible and relevant?

Design phase

5. *Ensure that security requirements are included in the design phase*.

Designers include security subjects and security objects using the notation of agile design diagrams. They incorporate the security classification of security subjects and objects to the modeling notation. They apply risk management as necessary to prioritize the features.

Implementation phase

6. *Ensure that the wanted security features are implemented*.

The security requirements and countermeasures to the threats must be implemented according to their security sensitivity. An implementation priority list is needed that indicates the priority of the functions to be implemented. The functions having a higher priority will be tested first.

Testing phase

7. *Test that the selected features work as wanted.*

Developers employ both use cases and abuse cases to check that the software satisfies selected (and designed) tests before the software is given to the customer. There must be a test list that indicates the priority of the cases to be tested. The features having a higher priority will be tested first.

4. AN ILLUSTRATION OF THE APPLICATION OF THE KEY SECURITY ELEMENTS

Abrahamsson *et al*. [1] review the existing agile methods, including Feature Driven Development [15], Internet-speed or short-cycle time systems development [6, 7] and extreme programming [8]. In the following example we illustrate how the key security elements are added into Feature Driven Development (FDD), described by Palmer and Felsing [15]. This agile method was chosen as an example of the application of the security process as it offers the most concrete modeling techniques for agile development [1].

4.1 Feature Driven Development

Feature Driven Development (FDD) consists of six phases [15].

1) Requirements analysis;
2) Develop an overall model;
3) Build a feature list;
4) Plan by feature;
5) Design by feature;
6) Build by feature.

Requirements analysis

According to FDD, developers capture requirements with use cases. In each use case, the actors are potential security subjects and the targets of the "use" (the assets that the actors access) are potential security objects (see Figure 1).

Primary actor/**security subject***:* User
Version: 1.0
Main success scenario:

User access document X

Security classification: confidential
Security objects: document X
Priority 1-10: 5

Figure 1. Textual use case scenario modified from [10] with security elements that are described in bold.

In Figure 1, the security elements (security subjects, security objects), as well as the version number and preliminary priority of this feature, are added to the use case description of Cockburn [10]. After sketching the security enriched use cases, the analyst lists potential abuse case scenarios. These can be divided into two categories.

The first category is unwanted actions carried out by the authorized persons (e.g., the employees of a company), and the use case scenarios reveal these authorized persons. For example, in the use case presented in Figure 1, we can sketch an abuse case scenario where the authorized user tries to access secret level documents. The second category of abuse cases is outsiders (see Figure 2).

Abuse case: Hacker
Version: 1.0
Possible abuse scenario: A hacker breaks in to the system and access to the customer file.
Frequency: several times a day
Abuse subject: a hacker
Security objects/target of abuse: customer file
Total costs of recovering: N USD
Priority 1-10: 5

Figure 2. An abuse case scenario modified from McDermott and Fox [13].

Figure 2 describes an abuse case scenario (hackers attempt to break into the system). It shows the assumed frequency of that action, presumed abuse subject, and the security object that is the target of the abuse. We also find the total costs of recovering from the abuse (N USD in Figure 2). Analysts need this "total costs of recovering" along with the frequency of the abuse scenario for prioritizing the abuse scenarios (Priority in Figure 2). At the requirements analysis phase, the abuse cases are at a high level of abstraction. Developers specify the abuse case scenarios during other phases of the development.

Develop an overall model

An overall model is derived from the use cases (cf., requirements analysis phase). First, candidate classes are made in the level of names only. The security levels of classes are derived from use cases (Figure 3).

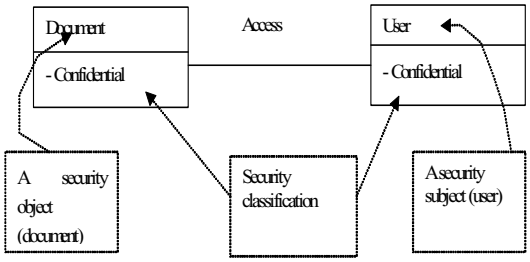

Figure 3. Listing of candidate classes and their security sensitivity, added to class diagrams of FDD [15].

Designers incorporate the security classification of the classes into the candidate classes (see Figure 3). In this way, the security classification is included informally within the object notation.

Build a feature list

In this phase, designers specify the abuse case scenario to include countermeasures to prevent the abuse case (i.e., the risk) from occurring (see Figure 4).

Abuse case: Hacker
Version: 1.0
Possible abuse scenario: A hacker breaks in to the system and gains access to the customer file.
Frequency: several times a day
Abuse subject: a hacker
Security objects/target of abuse: customer file
Total costs of recovering: N USD
Countermeasures: firewalls (only certain IP-addresses are allowed), encryption of passwords (at least 128-bit encryption), limit of login attempts per login name (three maximum attempts).
Number of the version/iteration where this abuse case is prevented: version 1.0
Priority 1-10: 10 - must be avoided!

Figure 4. An abuse case scenario modified from McDermott and Fox [13].

Designers may also need to determine at which iteration/version number, if any, is the point at which the software must be able to prevent the abuse case from occurring. For example, one may postpone the implementation of certain countermeasures for later versions due to e.g., time or budget limits. In Figure 5, we add the "Number of the version/iteration where this abuse case is prevented" element to the abuse cases.

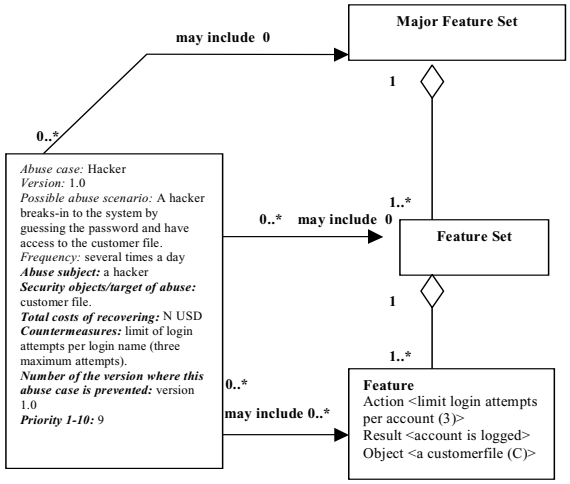

Figure 5. Security classifications and countermeasures to abuse scenarios are added to the feature list.

Figure 5 also presents the relationships of use cases and abuse cases to features. Use cases may include all features (major feature set, feature set, feature). Features are derived from use cases and abuse cases. The domain area may consist of one or more use cases, major feature set, feature set, and feature. The domain walk-through refers to the action of going through a certain area in the

software to be built. Abuse cases stand for a particular small area of the domain. Each feature belongs to only one feature set, and each feature set belongs to only one major feature set (area). Each feature includes <action>, <result> and <object>. Features are classified into sets. The feature set refers to a certain activity, such as a billing a service. The security elements can be added to the features as follows.

Designers incorporate abuse cases into the 'build a feature list' stage (Figure 5). Abuse cases in this stage include countermeasures on how the abuse case scenario is prevented from occurring. This prevention activity is also presented as a feature (to be implemented). To illustrate this, Figure 5 describes an abuse case where an abuse target is the customer file and countermeasures are described in the abuse case, including limit of log-in attempts to three per the same account. These security countermeasure features are as follows: Action is: <limit log-in attempts>, result is: <account is logged> and (security) object is <the customer file>. It is also possible to add security classifications of the security objects (e.g., customer file in Figure 5) to the feature. This security classification can be added to the end of the security object (cf., Figure 5). The aim of including the security classification in the feature list is to ensure that the security classification and security features do not disappear when moving between different phases. For example, in Figure 5, the customer file is labeled as confidential (C); see "Object <a customerfile (C)>" in Figure 5. The feature list is then described as a feature database (Table 2).

Table 2 describes the feature number (5 in this case), feature name (Limit of login attempts), and requirements cross-reference (Abuse case "hacker" priority: 9).

Table 2. Feature database table.

Feature nro	Feature name	Requirements Cross-ref
5	Limit of login attempts	Abuse case "hacker" priority: 9

Plan by feature

In this stage, designers put forth a development plan that guides the order of the features to be developed. They also prioritize security countermeasures (described as features) in order to decide the coding and testing order/priority of such features. Table 3 sketches this priority list.

Table 3. Security countermeasure feature priority list. In this table, 'priority' means priority number of a feature, 'nro' stands for a feature number, 'release number' is same as iteration number, and 'name' is the name of a feature. 'CP' refers to chief programmer, a term used by FDD method [15].

Priority	Nro	Release number	Name	CP	Date	Requirements Cross-ref
10	1	1	.....	...	....	.....
9	5	1	Limit of login attemp ts	John Doe	12/04	Abuse case "hacker" priority: 9

Table 3 lists security countermeasures listed in terms of their priority. Otherwise, the Security countermeasure feature priority list follows the notation of the Feature Database Table (cf., Table 2). The priority list also describes the iteration/release number (referring to the number of the iteration where this countermeasure must take place). CP is the name of the chief programmer who has the responsibility for the development of the feature. The date is the deadline of the feature development.

Design by feature

In this stage, we add the attributes to the classes. Figure 6 shows how we incorporate security classification into objects as attributes (e.g., Secclass: string).

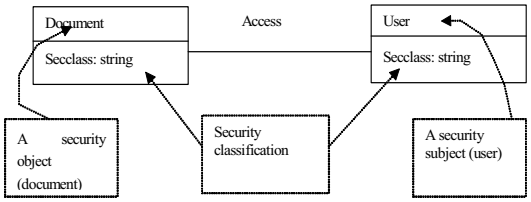

Figure 6. Adding security classification into the classes.

According to FDD, sequence diagram is also sketched in this stage (Figure 7).

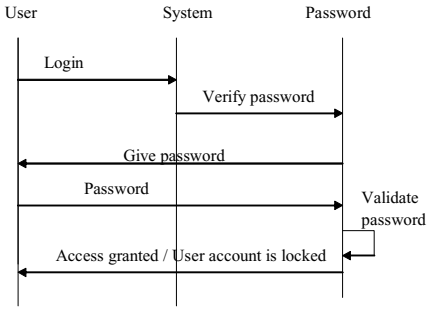

Figure 7. A sequence diagram.

Figure 7 present a sequence diagram representing a security feature: after three login attempts a user account is locked. In picture 7, a user refers to a person who wants to login to the system. The system represents a part of software that needs a login and password represents a class, which asks and validates passwords of the users. In this simple sequence diagram example, the user logs in to the system and the system calls password class to verify a password. In this case, password class asks for a password from the user (Give password) and user types in the password. Password class checks if the password is correct (validate password), and if the password is correct, then access is granted. If the password is incorrect the user account is locked, in our example.

Build by feature

In this stage the developers code and test the software. In the coding phase, they ensure that the security requirements are also included in the software to be built. In order to ensure that the security countermeasures are added according to their priority, developers follow the security countermeasure feature priority list in the coding phase. This procedure ensures that developers add the most important security features first when implementing the software. Later, developers can implement the security countermeasures having a lower level of priority are implemented, and so forth through the priority list. In the testing stage, developers test the selected abuse case scenarios starting with the cases that are deemed to be the most sensitive. This sensitivity priority is seen from the priority number of the abuse cases, and the testing is done following the security countermeasure feature priority list (see Table 3).

6. Summary

Agile software development methods generally lack specific software security features. While several separate security methods, such as checklists and management standards are possible, only a few of them can be seamlessly integrated to traditional software development methods. We believe that these security methods are even more difficult to integrate into agile software development. This article has provided an example of how security techniques can be added seamlessly to agile software development methods. We illustrated the addition of this solution to an agile development method, called Feature Driven Development. While further research and practical experiences are needed to test and fine-tune the approach presented in this paper, we see that our approach provides a promising solution to a critical problem in information systems and software development.

7. References

[1] Abrahamsson, P., Warsta, J., Siponen, M.T. & Ronkainen, J., (2003), New directions on agile methods: A comparative analysis. International Conference on Software Engineering.

[2] Backhouse, J., & Dhillon, G. (1996). Structures of responsibilities and security of information systems. European Journal of Information Systems. 5(1), 2-10.

[3] Baskerville, R., (1992), The Developmental Duality of Information Systems Security. Journal of Management Systems. Vol. 4, no. 1, pp. 1-12.

[4] Baskerville, R., (1993), Information Systems Security Design Methods: Implications for Information Systems Development. ACM Computing Surveys 25, (4) December, pp. 375-414.

[5] Baskerville, R., (2004), Agile security for information warfare: a call for research. Proceedings of the European Conferences on Information Systems (ECIS2004), 13-16 June, Turku, Finland.

[6] Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J., & Slaughter, S., (2003), Is Internet Speed Software Development Different? IEEE Software, Nov-Dec, pp. 70-78.

[7] Baskerville, R., & Pries-Heje, J., (2004), Short cycle time systems development. Information Systems Journal, Vol. 14 Issue 3.

[8] Beck K. (1999): Extreme Programming explained. Addison-Wesley.

[9] Booysen, H.A.S., & Eloff, J.H.P. (1995). A Methodology for the development of secure Application Systems. Proceeding of the 11th

IFIP TC11 international conference on information security.
[10] Cockburn, A., (2001), Writing Effective Use Cases. Addison-Wesley,
[11] Dhillon, G. and Backhouse, J., (2001), Current directions in IS security research: toward socio-organizational perspectives. Information Systems Journal. Vol 11, No 2.
[12] James, H.L. (1996). Managing information systems security: a soft approach. Proceedings of the Information Systems Conference of New Zealand. IEEE Society Press.
[13] McDermott, J. & Fox, C., (1999), Using abuse case models for security requirements, Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC). IEEE Computer Society Press.
[14] McDermott, J., (2003), Abuse-case-based assurance arguments. Proceedings 17th Annual Computer Security Applications Conference.
[15] Palmer S.R. and J.M. Felsing, (2002): A Practical Guide to Feature Driven Development. Prentice Hall.
[16] Pernul, G. (1992). Security constraint processing during multilevel secure database design. Proceedings of the 8th Annual Computer Security Applications Conference.
[17] Pernul, G. & Quirchmayr, G. (1994). Organizing MLS databases from a data modeling point of view. Proceedings of the 10th Annual Computer Security Applications Conference.
[18] Pernul, G., Tjoa A.M. & Winiwarter, W. (1998). Modeling Data Secrecy and Integrity. Data & Knowledge Engineering. 26, 291-308. North Holland.
[19] Saltmarsh, T.J., and Browne, P.S., (1983), Data Processing – Risk Assessment. In M. M. Wofsey (ed.): Advances in Computer Security Management, vol. 2, pp. 93-116, John Wiley and Sons Ltd.
[20] Sandhu, R., and Samarati, P., (1994), Access Control: Principle and Practice. IEEE Communications vol. 32, issue 9, pp. 40-48.
[21] Siponen, M.T., (2001), An analysis of the recent IS security development approaches: descriptive and prescriptive implications. In: G. Dhillon (eds:) Information Security Management - Global Challenges in the Next Millennium, Idea Group (2001).
[22] Siponen, M.T. and Baskerville, R. (2001)"A New Paradigm For Adding Security Into IS Development Methods" in Eloff, J., Labuschagne, L, von Solms, R. and Dhillon, G. (eds): *Advances in information security management & small systems security*. MA: Kluwer Academic Publishers.
[23] Straub, D.W, Welke, R.J, (1998), Coping with systems risk: security planning models for management decision making. MIS Quarterly, 22, 4, 441-64.
[24] Thomke, S., & Reinertsen, D. (1998). Agile product development: Managing development flexibility in uncertain environments. *California Management Review, 41*(1), pp. 8-30.
[25] Zurko, M.E, Simon, R.T, (1996), User-Centered Security. ACM New Security Paradigms Workshop, Lake Arrowhead, CA.