Integrating Security into Agile Software Development Methods

Tran Nguyen

Information System Analysis, Fall 2015

Dr. Sauter

Table of Contents

[1. Introduction 1](#_Toc435393736)

[2. The importance of integrating security into agile methodologies 1](#_Toc435393737)

[2.1. Overview about agile methodologies 1](#_Toc435393738)

[2.2. Overview of Information Security (InfoSec): 1](#_Toc435393739)

[2.3. Issues of integrating security into Agile Development Methodologies 1](#_Toc435393740)

[3. The suggested integration of security activities into Agile Methodologies 1](#_Toc435393741)

[3.1. Scrum 1](#_Toc435393742)

[3.1.1. Overview of Scrum 1](#_Toc435393743)

[3.1.2. The Proposed Secure Methodology for Scrum (Secure Scrum): 1](#_Toc435393744)

[3.1.3. Evaluation of Secure Scrum: 1](#_Toc435393745)

[3.2. Dynamic Systems Development Method 1](#_Toc435393746)

[3.2.1. Overview of Dynamic Systems Development Method 1](#_Toc435393747)

[3.2.2. The Proposed Secure Methodology for Dynamic Systems Development Method (Incorporating SQUARE steps into a DSDM process and Secure Dynamic System Development Method): 1](#_Toc435393748)

[3.2.3. Evaluation of “Incorporating SQUARE steps into a DSDM process” and “Secure Dynamic System Development Method”: 1](#_Toc435393749)

[3.3. Extreme Programming 1](#_Toc435393750)

[3.3.1. Overview of Extreme Programming 1](#_Toc435393751)

[3.3.2. The Proposed Secure Methodology for Dynamic Systems Development Method (Inbuilt Security and Role-based Extreme Programming) 1](#_Toc435393752)

[3.3.3. Evaluation of Inbuilt Security and Role-based Extreme Programming) 1](#_Toc435393753)

[4. Conclusion 1](#_Toc435393754)

[Table of Figures 1](#_Toc435393755)

[Bibliography 1](#_Toc435393756)

# Introduction

In the 1990s, in reaction to the heavyweight software development methods, many lightweight methods such as Extreme Programming, Dynamic Systems Development Method, Scrum and Crystal Clear were developed to be alternatives of the traditional method. In 2001, representatives from these lightweight methods unified and published the Agile Manifesto. Since then, Software Development Methods under the umbrella of Agile Manifesto have become popular. However, owing the fact that Agile methods has developed ten years ago, several new IT issues especially Information system security are not included in the framework. To avoid threats associated with information security, security needs to be integrated in Agile Software Development Methodologies.

This paper will explore techniques to integrate into Agile Software Development methods. In order to understand how the techniques work, the paper will first introduce the overview of agile methods, information security, scrum, dynamic systems development method and Extreme Programming. This will be followed by exploring the recent studies about integrating security into three agile methods (Scrum, Dynamic Systems Development Method and Extreme Programming). The paper will conclude with the evaluation of different techniques for integrating security into each agile method.

# The importance of integrating security into agile methodologies

## Overview about agile methodologies

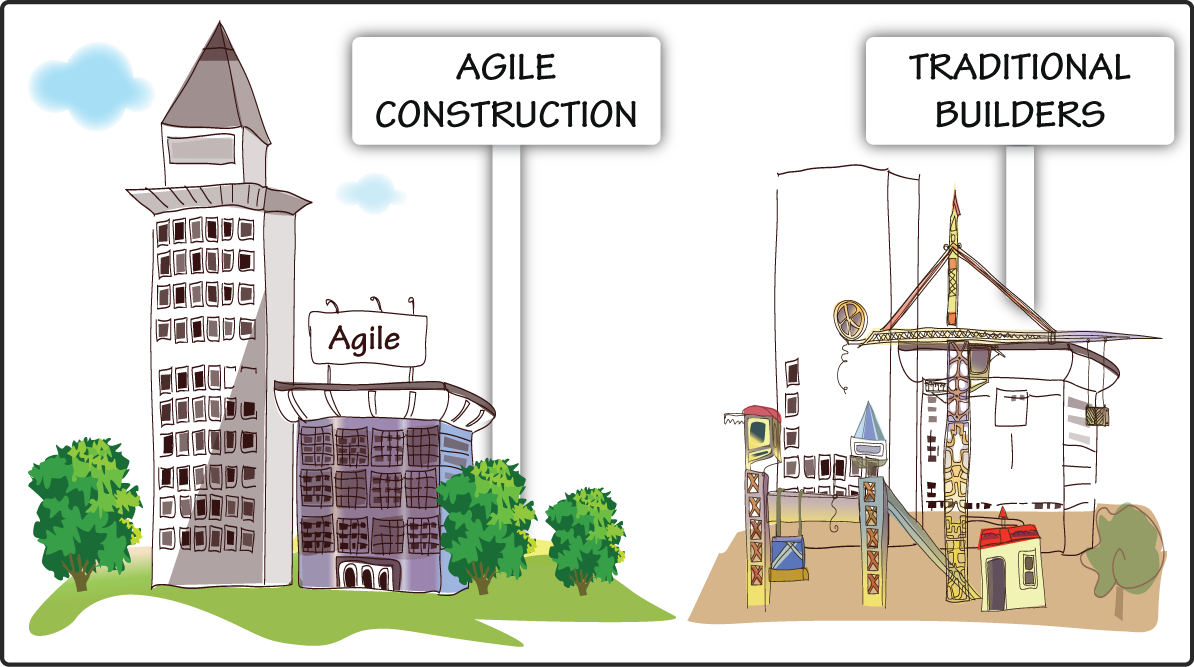
**Agile Methodologies** are software development methodologies that follow Manifesto for Agile Software Development. Agile Manifesto (Beck, et al., 2001) states that:

*“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

* *Individuals and interactions over processes and tools*
* *Working software over comprehensive documentation*
* *Customer collaboration over contract negotiation*
* *Responding to change over following a plan*

*That is, while there is value in the items on the right, we value the items on the left more.”*

The main difference between Agile methods and traditional waterfall method is the structure of the development. In the waterfall method, all product features and requirements follow a sequential design process. In the Agile development methods, the project is divided into iterations. Each iteration will implement a part of product requirements. The life cycle of the iteration will start over after each iteration is finished until all product requirements are satisfied.



Agile is a popular approach in Software Development. According to the Spring 2014 DDJ State of the IT Union Survey (Scott Ambler +Associates, 2014), compared to other development methodologies like Ad hoc, Lean and Traditional. Agile methodologies provide better results in return on investment, quality, stakeholder satisfaction, team morale, and delivery time. It is one of the three software development methodologies that have highest success rates (Scott Ambler +Associates, 2013).

There are various methodologies that apply the values of the Agile Manifesto. The popular agile methodologies are Scrum, Extreme Programming, Rational Unified Process, Dynamic System Development Method and Feature Driven Development. Each method or techniques has its own strengths and weaknesses. Depending on the characteristics of the system, the development team will choose an appropriate method or they can combine many methods.

<https://www.youtube.com/watch?v=OJflDE6OaSc>

## Overview of Information Security (InfoSec):

**Information Systems Security** is "the protection of information systems against unauthorized access to or modification of information, whether in storage, processing, or transit, and against the denial of service to authorized users, including those measures necessary to detect, document, and counter such threats" (Kissel, 2013).

To evaluate the information systems security, **C.I.A. triangle** is introduced as the classic industry standard for computer security since the development of the mainframe (Whitman & Mattord, 2014). the C.I.A triangle includes three attributes of information security: confidentiality, integrity and availability.

* Confidentiality: the information is protected from unauthorized individuals or systems.
* Integrity: the information remains whole, complete and uncorrupted.
* Availability: the information can be accessed by authorized users in the right format without interference or obstruction.

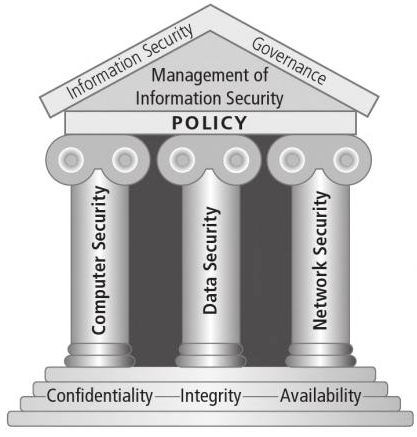


Figure 1 Components of Information Security (Whitman & Mattord, 2014)

If information security is described as a house, the C.I.A triangle is the foundation of the house. Components of information security such as Computer Security, Data Security and Network Security is built based on confidentiality, integrity and availability. In order to maintain a secure built system, Information Security Governance including the management of information security and policy needs to be implemented continuously.

## Issues of integrating security into Agile Development Methodologies

There are several issues that the Agile team have to overcome when integrating security into Agile Software Development:

* The pressure of short iteration (Bartsch, 2011) (Securosis, 2013). The short time of each iteration in Agile software development (only few weeks) is not enough to do necessary tests.
* Lack of information security knowledge (Securosis, 2013). Many programmers do not have enough knowledge about security problems. As a result, these programmers prefer to ignore security problems.
* Lack of security awareness (Bartsch, 2011). In point of fact, security issues need to be discussed when developing requirements. However, customers often do not concern about security because information security cannot help the project earn profit. It just protects the customers from losing their money.
* The compatibility of security activities and agile methodologies. Security Education and Awareness, Building Security Team, Static Code Analysis, Security Requirements Analysis and Reviewing Design Security are top five security activities that are compatible with Agile methodologies (Keramati & Mirian-Hosseinabadi, 2008). Threat Modelling is the biggest security-relevant challenge of Agile methodologies. Threat modeling is complex and it does not need customer interaction (Keramati & Mirian-Hosseinabadi, 2008).

# The suggested integration of security activities into Agile Methodologies

Although Agile Methodologies share common principles, each Agile Methodology has different ways to integrate security. This paper focuses on three agile software development methods: Scrum, Extreme Programming, and Dynamic Systems Development Method. Because only one or two techniques for integrating security introduced for each method, there are many techniques missing in this paper.

## Scrum

### Overview of Scrum

**Scrum** is a framework that uses fixed time-boxes of one month or less, called Sprint (Schwaber & Sutherland, 2013). When a Sprint end, the next Sprint will start until the product is finished. In Scrum framework, Scrum team will work through Scrum Events to achieve Scrum Artifacts.

Scrum Team consists of Product Owner, Scrum Master and Development Team.

* **Product Owner** is the person who is responsible for the result of the development.
* The **Development Team** is a group of people with different functional expertise working toward the system development.
* **Scrum Master** is responsible for ensuring Scrum Team work successfully and liaising with stakeholders outside the team. Scrum Master has no management authority.

In order to ensure every aspects of the process be visible to all Scrum Team member, Scrum Artifacts are designed and modified in each phase of process. Three common Scrum Artifacts are Product Backlog, Sprint Backlog and Burn down Chart. **Product Backlog** is an ordered list of requirements, desired functions and features of the product. Product Backlog is created from the early phase by Product Owner and it might be changed during the development process (Schwaber & Sutherland, 2013). From the Product Backlog, a set of selected items become **Sprint Backlog**. Sprint Backlog can be considered as the goal of the Sprint. After a task done or a Sprint done, the progress will be recorded in a **Burn down Chart** (James, 2012).

Scrum Team has to work through several events until all tasks in Sprint Backlog is reached. Typically, each Sprint has four events: Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective. **Sprint Planning** is when the Product Owner, Scrum Master and Development Team designs the Sprint Backlog and how they can get the work done. Every day, the Development Team has daily meeting facilitated by Scrum Master. In **Daily Scrum**, team members report what they did in the previous day, what their obstacles are and what they will do in that day. At the end of the Sprint, **Sprint Review** is held to demonstrate the working product increments. Before starting a new Sprint, **Sprint Retrospective** is a meeting for Scrum Master and Development Team to inspect themselves and create the improvement plan for the next Sprint (James, 2012).

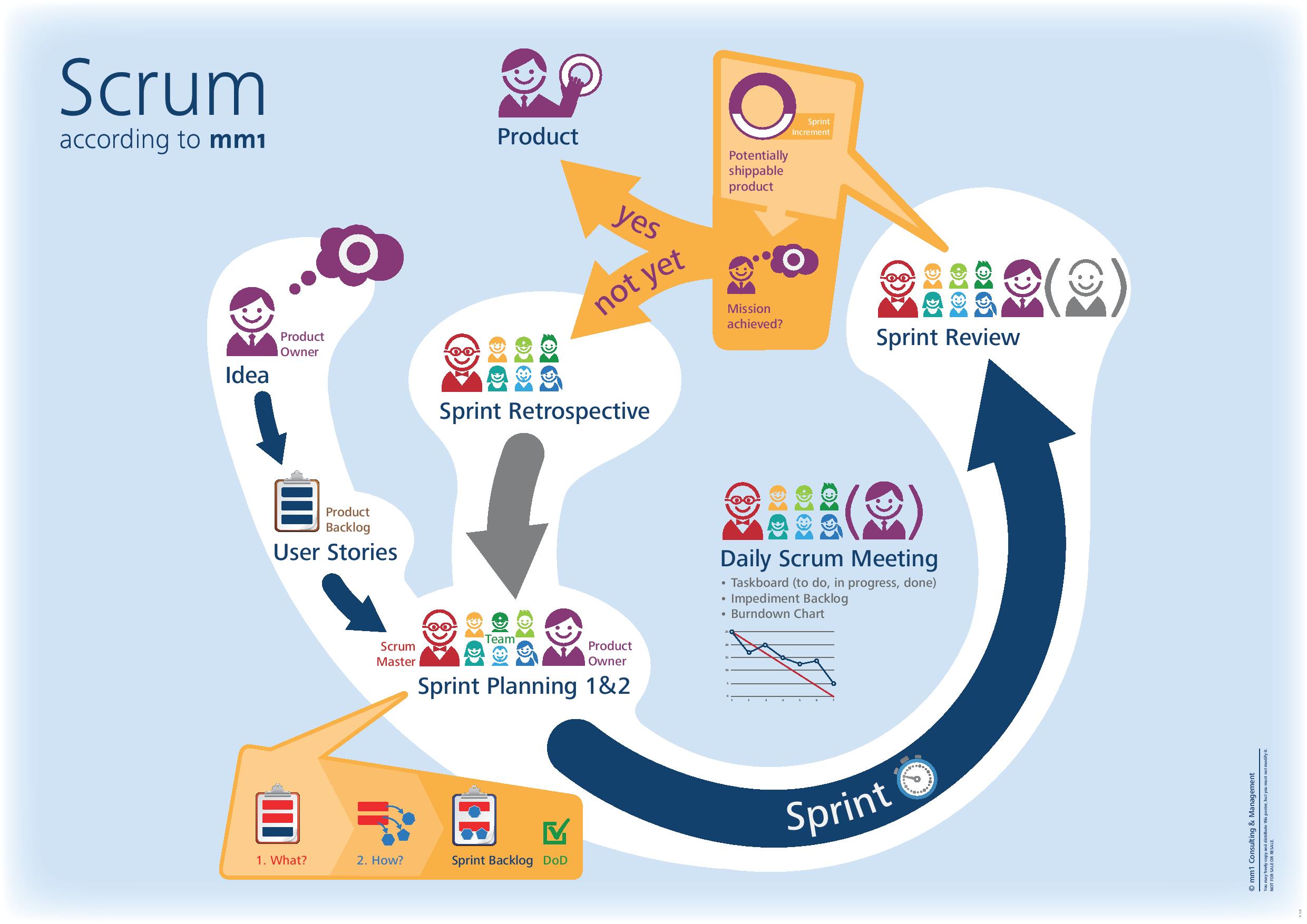


Figure 2: The Scrum Life Cycle (MM1, 2012)

### The Proposed Secure Methodology for Scrum (Secure Scrum):

There are several methodologies to integrate security into Scrum framework.

Veracode suggested two ways: The Security Sprint Approach and the Every-Sprint approach. In the Every-Sprint approach, the security user stories are included in every Sprint. The main problem of this method is the requirement of a security expert in the Scrum development team which is costly. In the Security Sprint approach, security user stories are analyzed and developed in a distinct Sprint. This approach might delay the development process.

Mongouei, Sani, and Almasi (2013) proposed a secure-enhanced version of Scrum, which is called S-Scrum. They added “Spikes” as a Scrum Events in Scrum Process. This version changed the simple structure of S-Scrum framework.

One of the lightest and simplest method is Secure Scrum. “Secure Scrum is a variation of the Scrum framework with special focus on the development of secure software throughout the whole software development progress” (Pohl & Hof, 2015).

Secure Scrum has four components: Identification, Implementation, Verification and Definition of Done. These four components will be integrated with six Scrum parts to increase the security of software Development. The heart of Secure Scrum is the use of S-Tag and S-Mark.

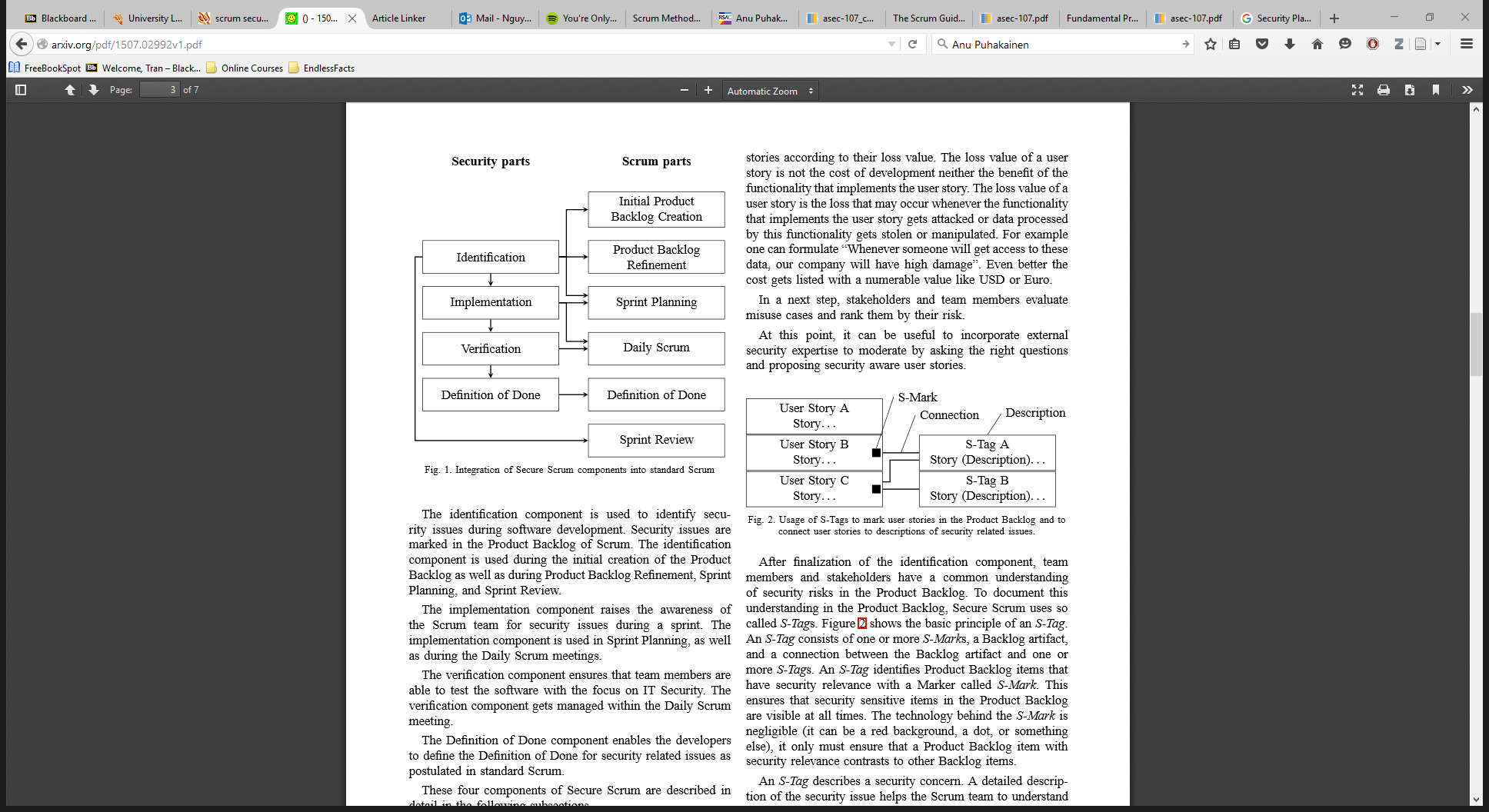


Figure 3 Integration of Secure Scrum components into standard Scrum (Pohl & Hof, 2015)

The **Identification Component** identifies security issues through the user stories of Product Owner and stakeholders. Then the security-relevant user stories are ranked by their risk and marked in the Product Backlog. The marker is called **S-Mark**, which can be a sticker, a dot or a color background. Based on marked user stories, a list of S-Tags is created. An **S-Tag** describes a security concern. An S-Tag might affect one or many Product Backlog Items. In order words, several Product Backlog items might share the same security concern.

Because Product Backlog Items can change over time to adapt the development process, S-tags can be modified when refining Product Backlog or planning new Sprint.

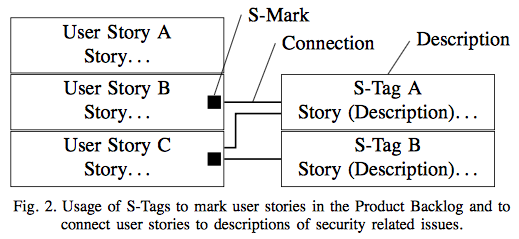


Figure 4 Usage of S-Tags to mark user stories in the Product Backlog and to connect user stories to descriptions of security related issues (Pohl & Hof, 2015)

The **Implementation component** is used to ensure the security awareness of the development team. So long as the user story is marked with an S-Mark, the S-Tag must be handled. In some cases, S-Tags can be divided into tasks. These tasks must be marked with an S-Mark.

The **Verification Component** is a part of the task. It ensures that the team member checked the security issues of S-Mark tasks. The verification component is a part of definition of done. The verification component is managed within the Daily Scrum meeting.

In some cases, the team members do not have enough knowledge or time to verify the security issues of the product and they need external resources. Thus the task cannot be done and a new task for verification is created. The new task will inherit S-Mark and all S-Tags from the original task. External resources can help Scrum Team in enhancing knowledge, solving challenges and providing external view in this task. The **Definition of Done** component ensure that the verification of the security has to be implemented by either internal or external resources.

### Evaluation of Secure Scrum:

The advantages of the Secure Scrum (Pohl & Hof, 2015) over the secure Scrum methods are:

* Increasing the security awareness of team members during the development process.
* Minimizing the cost of hiring IT security expert in the development team.
* Providing the external security resources when necessary.

The secure scrum solves two issues of integrating security into agile methods: lack of security knowledge and lack of security awareness. However, when identifying security issues, there is an absence of security expert. The problems related to security might not be recognized until the verification phase. The late presence of security expert can cause many problems such as the deficiency of security identification or the wrong direction while doing the security tasks.

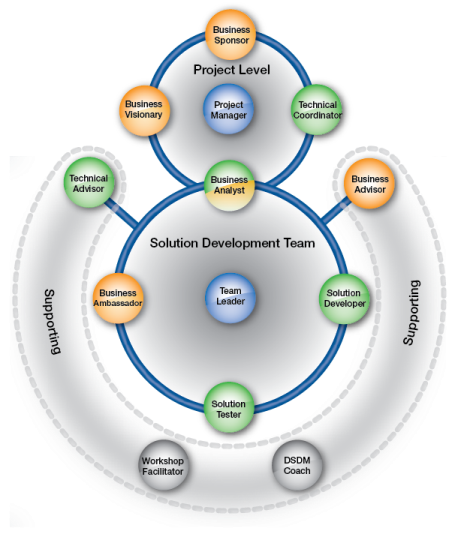
## Dynamic Systems Development Method

### Overview of Dynamic Systems Development Method

Dynamic Systems Development Method (DSDM) is an agile development framework that delivers the right solution at the right time (DSDM Consortium, 2015). DSDM is built based on Rapid Application Development. Thus, DSDM aims to minimize the time needed to develop the system.

In a DSDM Team, there are many roles related to the management view, the business view, the technical view and the progress view. A person can handle one or more roles. According to the newest version of the DSDM Agile Project Framework (DSDM Consortium, 2014), team roles are divided into three categories: Project-level Roles, Solution Development Team Roles and Supporting Roles.

* The **Project-level Roles** includes Business Sponsor, Business Visionary, Technical Coordinator, Project Manager and Business Analyst. They are responsible for the direction of the project.
* The **Solution Development Team Roles** are Business Ambassador, Solution Developer, Solution Tester, Business Analyst and Team Leader. Their responsibility is developing the system based on the direction of the Project-level Roles.
* The **Supporting Roles** do not need to engage with the project. They are Business Advisors, Technical Advisors, Workshop Facilitator and DSDM Coach. They assist the Solution Development Roles when necessary.



Orange: Roles representing the business view

Green: Roles representing the technical/solution view

Blue: Roles representing the management/leadership view

Grey: Roles representing the process view

Figure 5 Dynamic Systems Development Method Team Model (DSDM Consortium, 2014)

The Dynamic System Development Method has been developed and there are several differences between the original version and newer versions. Since the original model is widely used in practices and in theoretical research, this paper will use the Dynamic Systems Development Method life cycle version 4.2 developed by DSDM Consortium in 2007. This version is described as “Three pizzas and a cheese”. It has five phases: Pre Project, Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration, and Implementation (Stapleton, 1997).

The **Feasibility Study** phase is an assessment to check if the project should use Dynamic Systems Development Method approach or not. The purpose of **Business Study** is to understand the business processes affected and their information needs.

In the next two phases (the Functional Model Iteration and the Design and Build Iteration), there are four activities in each cycle: Identify Prototype, Agree Plan, Create Prototype and Review Prototype. The **Functional Model Iteration** focuses on building standard analysis models and product components. In the end of Functional Model Iteration, the Functional Prototype is produced. The **Design and Build Iteration** ensures that the system has sufficiently high standard and satisfies agreed requirements. The product of the Design and Build Iteration is the Tested System.

The **Implementation Phase** is a transition phase from development system to operational system. This phase includes several activities: training users, providing user guidelines, receiving the approval of user and reviewing business. The product of Implementation Phase is Delivered System and Increment Review Document.

Depending on the characteristics of the project, the Functional Model Iteration, Design and build Iteration, and the Implementation phase can be overlapped, rearranged or merged.

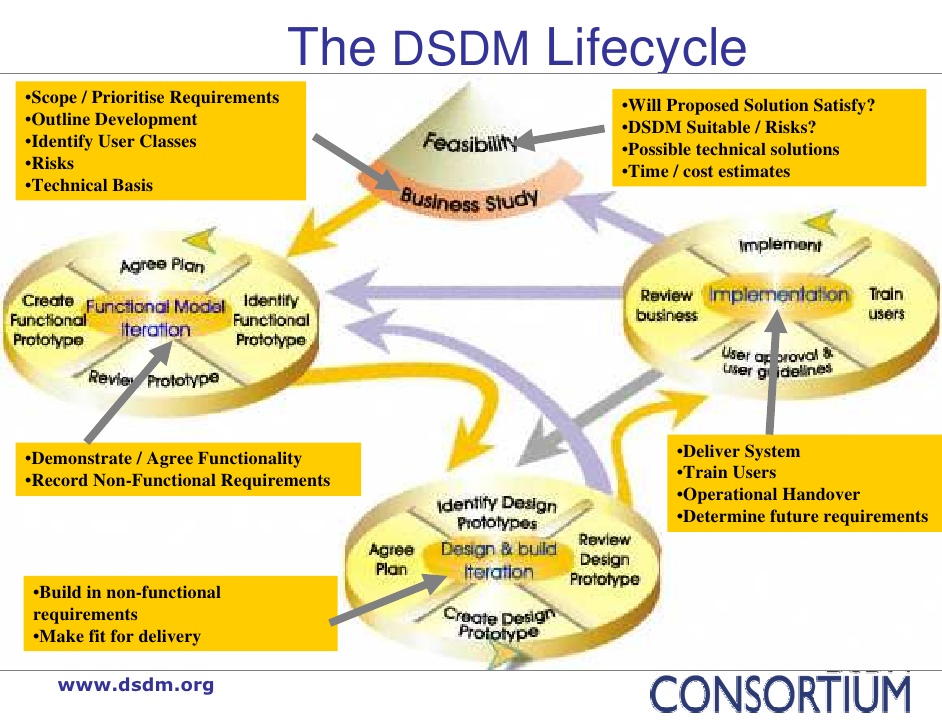


Figure 6 The DSDM Life Cycle (Messenger, 2006)

### The Proposed Secure Methodology for Dynamic Systems Development Method (Incorporating SQUARE steps into a DSDM process and Secure Dynamic System Development Method):

As other agile software development methodologies, Dynamic System Development Method do not pay attention to security issues. The research articles about the integration of security into DSDM are very limited. Based on the research on the presence of Software Security concerns in existing Agile practices including DSDM (Sani, Firdaus, Jeong, & Ghani, 2013), only one forum (DSDM Consortium 2002-2006) mentioned about security in Dynamic System Development Method.

One proposed technique is fitting SQUARE (Security Quality Requirements Engineering) into the Dynamic System Development Method framework (Mead, Viswanathan, & Zhan, 2008). SQUARE (System Quality Requirements Engineering) is a process that build up the security requirements for information system development. SQUARE Model has nine step and Med, Viswanathan, & Zhan (2008) has combined nine steps into Business Study and Functional Model Iteration of DSDM. The below table summarizes how SQUARE is incorporated into DSDM:

Table 1 Incorporating SQUARE Steps into a DSDM Process (Mead, Viswanathan, & Zhan, 2008)

|  |  |
| --- | --- |
| **DSDM Phase** | **SQUARE Step** |
| Business study | **Agree on definitions (Step 1)**  The requirements engineering team and project stakeholders agree on technical definitions that serve as a baseline for all future communication.  **Security goals (Step 2)** Security goals for the project are clearly outlined along with business goals  **Develop artifacts to support security requirements definitions (Step 3)** Create artifacts necessary for full understanding of the system.  **Perform risk assessment (Step 4)**  The initial risk assessment of RUP should also include security risk. The identified security risk should then be prioritized.  **Select elicitation technique (Step 5)** Select elicitation technique best suited for the project.  **Elicit security requirements (Step 6)**  Gather initial security requirements in terms of misuse cases that support the goals and security definitions already collected.  **Categorize requirements as to level (system, software) and whether they are requirements or other kind of constraints (Step 7)**  Categorize the elicited security requirements.  **Prioritize security requirements (Step 8)**  The categorized security requirements are prioritized. Prioritizing techniques such as Analytical Hierarchical Process can be used.  **Requirements inspection (Step 9)**  The second milestone review process of RUP should include inspection of the security requirements. |
| Functional model iteration | **Perform risk assessment (Step 4)**  This step is revisited to develop a revised risk list. Additional security risks are captured and risks are prioritized. **Perform Steps 6, 7, 8, and 9 if new risks arise out of Step 4.** |

The limitation of this technique is focusing on only security requirements in the early phases of DSDM (Business Study Phase).

On the contrary with the “Incorporating SQUARE steps into a DSDM process” technique, Sani, Shani, and Jeong (2013) added security phases and sub-phases in later phases of DSDM framework. They called the extended framework as Secure Dynamic Systems Development Method (SDSDM). There are four extensions in SDSDM:

* Adding new sub-phase “Identify security concerns/issues” in Functional Model Iteration phase
* Adding new phase “Secure Design”
* Adding new phase “Secure Functional Model Iteration”
* Adding some security elements in Implementation phase

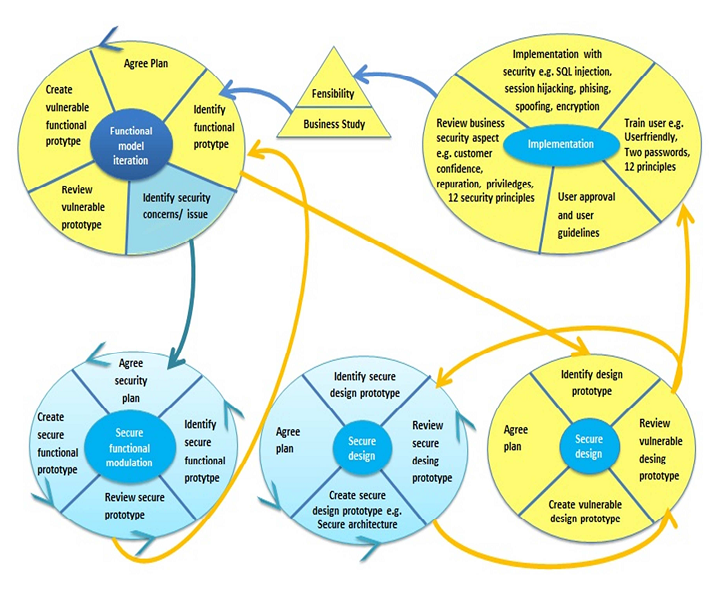


Figure 7 Secure Dynamic Development Method Life Cycle (Sani, Shani, & Jeong, Secure Dynamic System Development Method (SDSDM): Model for Secure Software Development, 2013)

Initially, security issues are discussed in the general functional model iteration. Identifying the functional prototype and identifying security concerns are performed simultaneously.

Then, two phases – secure design phase and secure functional model are added into the development process. Secure functional model phase is performed after functional model iteration and secure design is performed after Design and Build phase. Basically, the **Secure Design** phase and **Secure Functional Model** Iteration phase are the Functional Model Iteration and Design and Build Iteration which focus on security activities. These two new phases have the same structure with 4 sub-phases: Identify Prototype, Agree Plan, Create Prototype, and Review Prototype.

Because running a secure system needs professional knowledge, training courses for users are highly encouraged in implementation phase. When using the system, the users will review the security of the system and help the developers recognize security issues.

### Evaluation of “Incorporating SQUARE steps into a DSDM process” and “Secure Dynamic System Development Method”:

SQUARE is a process model for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications. By incorporating SQUARE into DSDM, the security requirements are built comprehensively in Business Study. Moreover, the security requirements are revised in functional model iteration. As a result of this method, the security requirements are fully developed. Unfortunately, security requirements cannot ensure the implementation of security tasks in the later phases.

On the contrary with “Incorporating SQUARE steps into a DSDM process”, the Secure Dynamic System Development Method add security-relevant elements into later phases of DSDM. Secure Dynamic System Development method does not discuss details about analyzing the security issues. With two new phases “Secure Design” and “Secure Functional Model”, security codes are built and tested during the development process. The innovative features of the Secure Dynamic System Development Method are security training and security review in implementation phase. Since security tests cannot be handled in short time of iteration, the help of users during implementation is a solution to find errors and improve the security of products.

“Incorporating SQUARE steps into a DSDM process” and “Secure Dynamic System Development Method” can combine together to create a framework for integrating security into DSDM. First, security requirements will be developed in the business study phase and revised in Function model iteration. Then security tasks will be built, tested and reviewed by Secure Dynamic System Development Method.

## Extreme Programming

### Overview of Extreme Programming

Extreme Programming is a lightweight methodology developing software in the face of vague or rapidly changing requirements based on addressing constraints in software development (Beck & Andres, Extreme Programming Explained: Embrace Change, 2004).

There are a number of roles in Extreme Programming team: programmer, customer, tester, tracker, coach, consultant, and big boss (Beck, Extreme Programming Explained: Embrace Change, 1999). In these roles, programmer, customer, coach and tracker are must-have roles in the team. A person can have more than one role but they have to be aware which hat they are wearing.

* The **programmer** is a person working directly with program. They can be the only technical people in the team. In Extreme Programming, the programmer has to communicate with other people including technical people and business people.
* The **customer** is the one who know what to program. Unlike customers in other frameworks, an Extreme Programming customer have to learn skills like stories writing skill, functional test skill and decision making skill.
* The **tracker** is the conscience of the team. They monitor the working progress and give feedback.
* The **coach** is a person who are responsible for the process as a whole. The coach will guide the team to work better but ensure that the team can work independently.

There are twelve major practices in Extreme Programming (Beck, Extreme Programming Explained: Embrace Change, 1999).

* The **Planning Game**: The scope of the next release is identified by combining business requirements (Scope of the project, priority of work items, composition of releases, and dates of releases) and technical requirements (Time estimation, technical consequences, process, and detailed scheduling)
* **Small Releases**: Each releases should be as small as possible and their cycle should be as short as possible.
* **Metaphor:** Metaphor is a guidance for the system development. It shows how the whole system works.
* **Simple Design**: The design of the system should be as simple as possible. The design should run all the tests, have no duplicated logic, states every important intention, and have the fewest possible classes and methods.
* **Testing**: The unit test by programmers and the functional test by customer are required for each program feature.
* **Refactoring**: When implementing a program feature, the programmers find ways to make the program simpler. The simpler design helps programmer work better on the next feature.
* **Pair Programming**: All production code is written with two programmers at one machine. One person works directly and the other person evaluate and improve the code.
* **Collective Ownership**: Anyone can change any code anywhere in the system at any time. This means everyone takes responsibility for the whole of the system.
* **Continuous Integration**: Every time a task is completed, the programmers have to integrate their changes into the system and run the test.
* **40 Hour Week**: Overtime working is highly discouraged. Working more than 60 hours a week for many weeks can reduce the creativity, the carefulness and the confidence of team members.
* **On-site Customer**: There should be a real user on the team. They are available to answer all questions and help to build a successful system.
* **Coding Standards**: Programmers follow the rules of code and communication.

The twelve practices support each other. Each practice is described as a puzzle piece and Extreme Programming is a jigsaw puzzle.

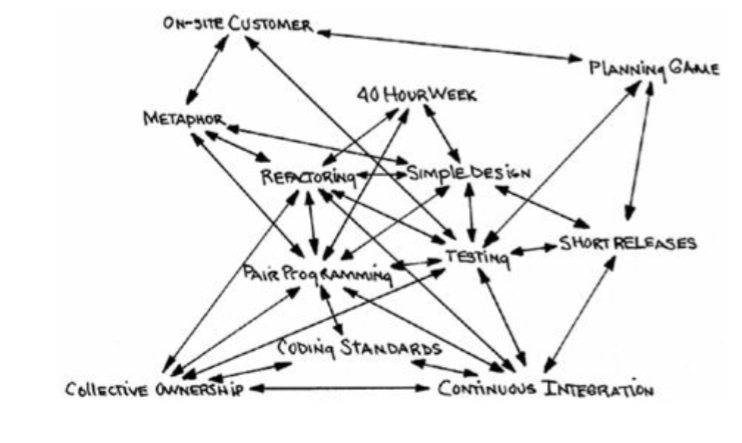


Figure 8 The Extreme Programming Practices support each other (Beck, Extreme Programming Explained: Embrace Change, 1999)

### The Proposed Secure Methodology for Dynamic Systems Development Method (Inbuilt Security and Role-based Extreme Programming)

According to the literature review about Software Security Engineering in Extreme Programming Methodology (Ghani & Yasin, Software Security Engineering in Extreme Programming Methodology: A systematic Literature Review, 2013), although some researchers do agree that security elements can be integrated into Extreme Programming, there are few specific research studies about secure Extreme Programming. Two notable research studies about integrating security into Extreme Programming are “Role-based Extreme Programming (XP) for Secure Software Development” (Ghani, Izzaty, & Firdaus, Role-based Extreme Programming (XP) for Secure Software Development, 2013) and “Improved Extreme Programming Methodology with Inbuilt Security” (S., Norwawi, Selamat, & Sharif, 2011).

According to Ghani et al. (2013), adding security elements into Extreme Programming framework without a security expert is not effective. They introduced a new role in Extreme Programming called “Security Master”. A **Security Master** is responsible for the security of the developed software. Fundamentally, a Security Master is a Programmer who focus on security elements.

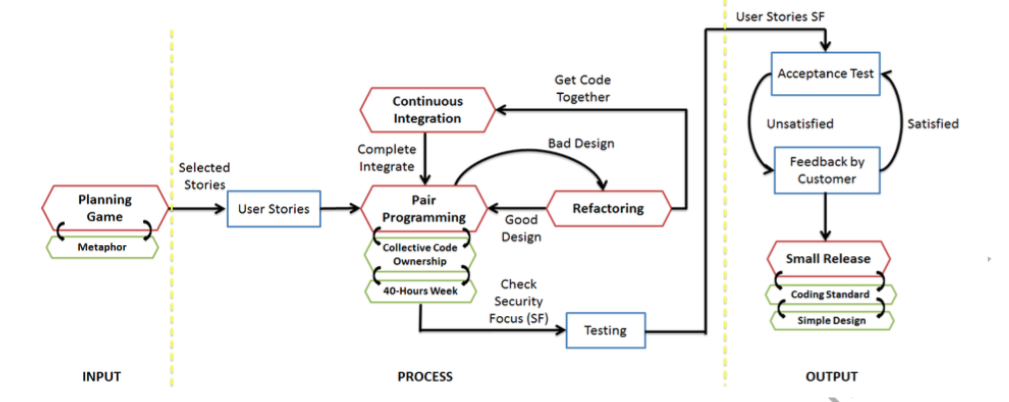


Figure 9 Ten Extreme Programming Practices of a Security Master (Ghani, Izzaty, & Firdaus, Role-based Extreme Programming (XP) for Secure Software Development, 2013)

The Security Master involves in ten Extreme Programming Practices: planning game, metaphor, coding standard, simple design, small release, continuous integration, pair programming, collective code ownership, 40-hours per week and refactoring. Figure 8 summarizes the working progress of a Security Master. First of all, the Security Master lists security requirements and adds security elements into the metaphor. Like other programmers in Extreme Programming, they work 40 hours a week, program in pair, get the code together every time the task completed. If the security design if not good enough, they have responsibility to refactor. Different from programmers who do unit test and customers who do functional test, the Security Master has to do security test in Testing Phase.

Inbuilt Security is another way to integrating security into Extreme Programming (S., Norwawi, Selamat, & Sharif, 2011). The method behind the Inbuilt Security is adding security-relevant elements into eleven out of ten practices of the Extreme Programming framework:

* Whole Team (On-site Customer): a customer in extreme programming are not only a purchaser but also a user. The customer will help Extreme Programming Team list misuse cases and review security policies.
* Planning Game: Business people consider security issues and redefine the system requirements. Based on the security-relevant system requirements, the team will split task into smaller units.
* Simple Design: Stating the intention of the programmer clearly is a part of simple design. For that reason, the security specification should be clarified.
* Design Improvement (Refactoring): Executing risk assessment for vulnerability is one way to revise the products and improve the design.
* Coding Standards: Standardizes codes in security ensure that the other practice of DSDM – Collective Code Ownership is able to achieve.
* Pair Programming: Like other functions, security codes have many errors and working in pair can reduce the possibility of errors.
* Collective Code Ownership: To support others to edit the code later, the standardized and immune codes are used in security alignment.
* Metaphor: In the metaphor, functions that have security implications are desired to notice and separate.
* Test-Driven Development (Testing): Test and feedbacks on vulnerabilities give some level of security assurance. Automated tool is used for security compliance analysis and portfolio analysis.
* Continuous Integration: Integrating with the system help the programmer recognize errors including security errors and fix it soon.
* Small Release: small release enable customer and programmers control the fulfillment of security requirement of each release.

### Evaluation of Inbuilt Security and Role-based Extreme Programming)

Inbuilt Security and Role-based Extreme Programming have different approaches. Both of the methods keep the core concepts of Extreme Programming. The development process of Extreme Programming is unchanged.

Inbuilt Security method emphasizes the insertion of security-relevant tasks into the practices of Extreme Programming. This method does not mention about new team member. The extreme programming team without a security expert can prevent the team from the accomplishment of security tasks. Furthermore, some practices of extreme programming conflict with the requirements of security activities. For example, the involvement of business people in threat modelling can cause an overload of jargon and reduce the effectiveness of threat modelling.

Role-based Extreme Programming draws attention on a new role in Extreme Programming called Security Master. The security of the product heavily depends on the skills of the Security master. If the security master is unqualified, the hole in the security of the product is unavoidable.

# Conclusion

As the number of software hit by cyber attacks are skyrocketing, building security into the software development become an essential part of software development processes. Agile approach is one of the most popular software development approach and methodologies using this approach needs to be modified to adapt the increase threats and vulnerabilities of information system. Many researchers around the world are developing secure agile methodologies based on the existing Agile methodologies. This paper introduced Secure Scrum, Secure Dynamic System Development Method, SQUARE in Dynamic System Development Method, Role-based Extreme Programming, and Inbuilt Security in Extreme Programming. The Secure Scrum suggests security marks and tags on user stories, backlogs and tags. The Secure Dynamic System Development Method offers new phases and sub-phases into Dynamic System Development Method Life Cycle. SQUARE in Dynamic System Development Method proposes SQUARE steps included in business study phase. Role-based Extreme Programming recommends a new team member – security master. Inbuilt Security in Extreme Programming adds security-relevant elements into Extreme Programming practices. Although several issues of integrating security into agile methodologies are solved, those methods have many limitations. The combination of related methods can eliminate some weaknesses and improve the existing methods.

# Table of Figures

[Figure 1 Components of Information Security (Whitman & Mattord, 2014) 1](#_Toc435393477)

[Figure 2: The Scrum Life Cycle (MM1, 2012) 1](#_Toc435393478)

[Figure 3 Integration of Secure Scrum components into standard Scrum (Pohl & Hof, 2015) 1](#_Toc435393479)

[Figure 4 Usage of S-Tags to mark user stories in the Product Backlog and to connect user stories to descriptions of security related issues (Pohl & Hof, 2015) 1](#_Toc435393480)

[Figure 5 Dynamic Systems Development Method Team Model (DSDM Consortium, 2014) 1](#_Toc435393481)

[Figure 6 The DSDM Life Cycle (Messenger, 2006) 1](#_Toc435393482)

[Figure 7 Secure Dynamic Development Method Life Cycle (Sani, Shani, & Jeong, Secure Dynamic System Development Method (SDSDM): Model for Secure Software Development, 2013) 1](#_Toc435393483)

[Figure 8 The Extreme Programming Practices support each other (Beck, Extreme Programming Explained: Embrace Change, 1999) 1](#_Toc435393484)

[Figure 9 Ten Extreme Programming Practices of a Security Master (Ghani, Izzaty, & Firdaus, Role-based Extreme Programming (XP) for Secure Software Development, 2013) 1](#_Toc435393485)

# Bibliography

Bartsch, S. (2011). Practitioners’ Perspectives on Security in Agile Development. *2011 Sixth International Conference on Availability, Reliability and Security.* IEEE.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change* (1st ed.). Addision-Wesley.

Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Addition-Wesley.

Beck, K., Beedle, M., Bennekum, A. v., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001). Retrieved from AgileManifesto.Org: http://www.agilemanifesto.org/

DSDM Consortium. (2008). *Overview of DSDM version 4.2*. Retrieved from DSDM Web site: http://www.dsdm.org/version4/2/public/Overview\_of\_DSDM.asp

DSDM Consortium. (2014). *The DSDM Agile Project Framework*. Retrieved from DSDM Web site: http://www.dsdm.org/dig-deeper/book/7-roles-and-responsibilities

DSDM Consortium. (2015). *What is DSDM?* Retrieved from DSDM web site: http://www.dsdm.org/content/what-dsdm

Ghani, I., & Yasin, I. (2013, April). Software Security Engineering in Extreme Programming Methodology: A systematic Literature Review. *Science International, 25*(2), 215.

Ghani, I., Izzaty, N., & Firdaus, A. (2013, January). Role-based Extreme Programming (XP) for Secure Software Development. *Science International*, 1071.

James, M. (2012). *Scrum Reference Card.* Retrieved from CollabNet Web site: http://www.collab.net/sites/default/files/uploads/CollabNet\_scrumreferencecard.pdf

Keramati, H., & Mirian-Hosseinabadi, S.-H. (2008). Integrating Software Development Security Activities with Agile Methodologies. *2008 IEEE/ACS International Conference on Computer Systems and Applications* (pp. 749 - 754). IEEE.

Kissel, R. (2013, May). *Glossary of Key Information Security Terms.* Retrieved from National Institude of Standards and Technology: http://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf

Mead, N. R., Viswanathan, V., & Zhan, J. (2008). Incorporating Security Requirements Engineering into Standard Lifecycle Processes. *International Journal of Security and its Applications, 2*(4), 67-79.

Messenger, S. (2006). *Introduction to DSDM* . Retrieved from Slideshare: http://www.slideshare.net/nashjain/introduction-to-dsdm

MM1. (2012, May 21). *Scrum According to MM1*. Retrieved November 5, 2015, from MM1 Consulting and Management Website: http://mm1.com/scrum-poster-for-easy-implementation-of-scrum-published-by-mm1-consulting-management/

Mougouei, D., Sani, N. F., & Almasi, M. M. (2013). S-Scrum: a Secure Methodology for Agile Development of Web Services. *World of Computer Science and Information Technology Journal (WCSIT), 3*(1), 15-19.

Plaza, L. (2015, January 21). *Difference between Product Backlog and Sprint Backlog*. Retrieved from Management Plaza Company Web site: http://mplaza.pm/difference-between-product-backlog-and-sprint-backlog/

Pohl, C., & Hof, H.-J. (2015). Secure Scrum: Development of Secure Software with Scrum. *arXiv preprint arXiv:1507.02992*.

S., B. M., Norwawi, N. M., Selamat, M. H., & Sharif, K. Y. (2011). Improved Extreme Programming Methodology with Inbuilt Security. *2011 IEEE Symposium on Computers & Informatics*, (pp. 674-679).

Sani, A., Firdaus, A., Jeong, S. R., & Ghani, I. (2013, May). A Review on Software Development Security Engineering using Dynamic System Method (DSDM). *International Journal of Computer Applications, 69*(25).

Sani, A., Shani, I., & Jeong, S. R. (2013). Secure Dynamic System Development Method (SDSDM): Model for Secure Software Development. *Journal of Science International Lahore, Special Issue*, 1059-1064.

Schwaber, K., & Sutherland, J. (2013, July). *The Definitive Guide to Scrum: The rules of the Game.* Retrieved from The Scrum Guide: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100

Scott Ambler +Associates. (2013). *2013 IT Project Success Rates Survey Results*. Retrieved November 13, 2015, from Ambysoft: http://www.ambysoft.com/surveys/success2013.html

Scott Ambler +Associates. (2014). *Software Development at Scale: Results from the Spring 2014 DDJ State of the IT Union Survey*. Retrieved November 13, 2015, from Ambysoft: http://www.ambysoft.com/surveys/stateOfITUnion2014Q2.html#Results

Securosis. (2013, November 3). *Secure Agile Development.* Retrieved November 12, 2015, from Securosis, L.L.C.: https://securosis.com/assets/library/reports/SecureAgileDevelopment\_Nov2014\_FINAL.pdf

Stapleton, J. (1997). *DSDM, Dynamic Systems Development Method: The Method in Practice.* Addison-Wesley.

Veracode. (2010). *Agile Security: Successful Application Security Testing for Agile Development*. Retrieved November 5, 2015, from https://www.veracode.com/sites/default/files/Resources/Whitepapers/whitepaper-agilesecurity.pdf

Whitman, M., & Mattord, H. (2014). *Principles of Information Security* (5th Edition ed.). Cengage Learning.