Validating Security Design Pattern Applications Using Model Testing

Takanori Kobashi
Waseda University
Computer Science and Engineer Department
Tokyo Japan
kobashi@akane.waseda.jp

Nobukazu Yoshioka
GRACE Center
National Institute of informatics
Tokyo Japan
nobukazu@nii.ac.jp

Takao Okubo
Information Security Division
Institute of Information Security
Yokohama Japan
okubo@iisec.ac.jp

Haruhiko Kaiya
Architecture Research Division
National Institute of Informatics
Tokyo Japan
kaiya@nii.ac.jp

Hironori Washizaki
Waseda University
Computer Science and Engineering
Tokyo Japan
washizaki@waseda.jp

Yoshiaki Fukazawa
Waseda University
Computer Science and Engineering
Tokyo Japan
fukazawa@waseda.jp

***Abstract*— Software developers are not necessarily security specialists, security patterns provide developers with the knowledge of security specialists. Although security patterns are reusable and include security knowledge, it is possible to inappropriately apply a security pattern or that a properly applied pattern does not mitigate threats and vulnerabilities. Herein we propose a method to validate security pattern applications. Our method provides extended security patterns, which include requirement- and design-level patterns as well as a new model testing process using these patterns. Developers specify the threats and vulnerabilities in the target system during an early stage of development, and then our method validates whether the security patterns are properly applied and assesses whether these vulnerabilities are resolved.**

Keywords-component; Security Patterns; Model Testing; Test-Driven Development; UML;

I. INTRODUCTION

Due to the increased number of business services on open networks and distributed platforms, security has become a critical issue. Developers must support software with security measures [1]. However, security concerns must inform every phase of software development from requirements engineering to design, implementation, testing, and deployment. Due to the vast number of security concerns and the fact that not all software engineers are security specialists, creating software with adequate security measures is extremely difficult.

Patterns are reusable packages that incorporate expert knowledge. Specifically, a pattern represents a frequently recurring structure, behavior, activity, process, or "thing" during the software development process. To resolve security issues, many security design patterns have been proposed [2], [3]. For example, reference [2] shows 25 design-level security patterns.

Currently, threats and vulnerabilities within a system are insufficiently identified during the early development stage. Although UML-based models are widely used for design, in model-driven software development, the appropriateness of the security patterns or whether the model for the applied patterns satisfies the security requirements is often not validated [1]. It is possible to apply a security pattern inappropriately. Additionally, properly applying a security pattern does not guarantee that threats and vulnerabilities are mitigated. These issues may cause security damage. Thus, our research aims to answer two Research Questions (RQs).

- **RQ1**: Can an appropriate application of the security design pattern in a design model be validated?
- **RQ2**: In a design model, can the existence of vulnerabilities identified at the requirement stage be validated before and after pattern application?

Herein we answer these two questions. Because the Security Pattern alone does not provide systematic guidelines with respect to applications, we propose validating security design pattern applications using model testing in an UML [4] model simulation environment. The method consists of two validations in the design phase: one validates whether the security patterns are appropriately applied and the other validates whether the design model–applied patterns are vulnerable to the threats identified in the requirement stage. Our method provides two major contributions:

- New extended security patterns, which include requirement- and design-level patterns
- A new model-testing process using these extended patterns

This paper is organized as follows. Section II describes the background and problems with security software development. Section III details our new method, which integrates the security patterns. Section IV applies our pattern to a case study. Section V describes potential weaknesses of our method. Finally, Section VI summarizes this paper.

II. BACKGROUND AND PROBLEMS

A. UML extension for modeling security constraints

UML-based models have been recently used for design. In particular, UMLsec [5] and SecureUML [6] have been

proposed to address security concerns. UMLsec is defined in the form of a UML profile using standard UML extension mechanisms. Stereotypes with tagged values are used to formulate the security requirements, and then the constraints are used to verify whether the security requirements hold during specific types of attacks. However, developers who are not security specialists have difficulty in employing UMLsec and must receive special training, which involves both time and money.

B. *Security Requirement Patterns*

The security requirement pattern is an existing technique to identify assets, threats, and countermeasures [7]. A security pattern is reusable as a security package and includes security knowledge, allowing software developers to design secure systems like a security expert. Various types of security patterns exist. For example, the security requirement pattern (SRP) is used at the requirement level, while the security design pattern, which is described in Section C, is applied at the design stage level.

The "Structure" of SRP uses the Misuse case with the Assets and Security Goal (MASG) model [8], which is an extension of the misuse case [9] that provides the structure of assets, threats, and countermeasures at the requirement level. This enables developers to model attackers, attacks, and countermeasures as well as normal users and their requirements. In addition to the elements of misuse case diagrams, the MASG model consists of the following elements:

- Data assets: Assets to be protected
- Use case assets: Functions related to assets
- Security goals: Reasons to protect assets

Identifying assets improves threat recognition, while identifying security goals determines what security measures are important in the target system. The MASG model also contains a security requirement analysis process. First, the assets of the system are identified, and the security goals are defined. Next, threats that may violate the goals are defined, and security countermeasures against these threats are determined [7]. Finally, the security countermeasures that satisfy the security goals are confirmed.

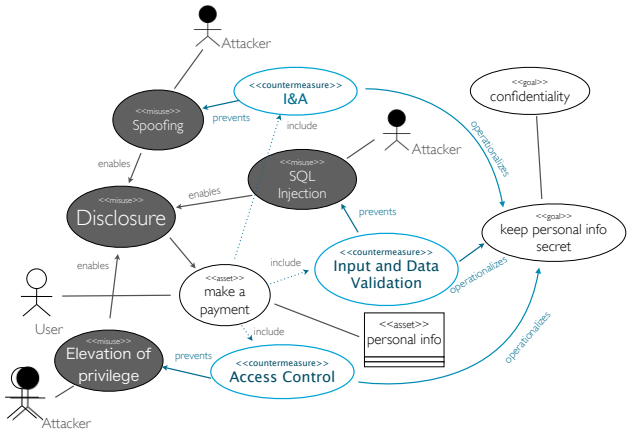

Figure 1. Sample MASG model for a shopping website

Figure 1 shows a typical example of a MASG model: a partially modeled shopping website. The function "make a payment" has several assets, which could be threatened. In the model, "Disclosure" is a threat for "make a payment", while "personal information" is an asset. "Spoofing", "Elevation of privilege", and "SQL Injection" enable Disclosure. In addition, each countermeasure, such as "Identification and Authentication (I&A)", "Access Control", or "Input and Data Validation", effectively mitigate these threats. Although the MASG model comprehensively explores security issues at the requirement level, it does not determine whether the identified threats actually exist in the design model.

C. *Security Design Patterns*

To satisfy security specifications, the use of Security Design Patterns (SDPs) is an established technique. The SDP includes "Name", "Context", "Problem", "Solution", "Structure", "Consequences", and "See Also". The pattern descriptions can be reused in multiple systems.

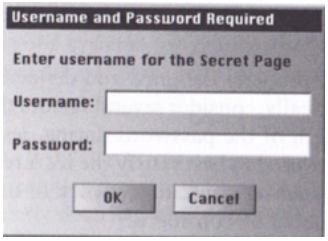

Figure 2. Structure of SDP (Password Design and Use pattern)

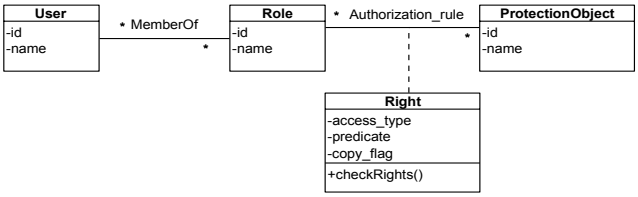

Figure 3. Structure of SDP (RBAC pattern)

Figures 2 and 3 show examples of the SDP structure. The Password Design and Use pattern describes the best security practice to design, create, manage, and use password components to support the I&A requirements. In addition to configuring or managing passwords, engineers and administrators use password constraints to build or select password systems. The RBAC pattern, which is a representative pattern for access control, describes how to assign precise access rights to roles in an environment where access to computing resources must be controlled to preserve confidentiality and the availability requirements.

D. *Motivating example*

As an example of a pattern application, Fig. 4 shows a portion ("make a payment") of a UML class diagram to realize a payment process on the Web. A SDP alone is insufficient to

support the development lifecycle because it lacks systematic guidelines with respect to applications in the entire lifecycle [10]. Consequently, formally describing what rules must be verified is difficult [11]. In addition, most SDPs do not specifically mention the systematic guidelines until the relations with the Security Requirement are defined [1]. Under the present conditions, it is possible that a developer may inappropriately apply a security measure to an identified threat. Additionally, the appropriateness of the applied pattern to the model and the pattern's ability to resolve vulnerabilities are inadequately verified.

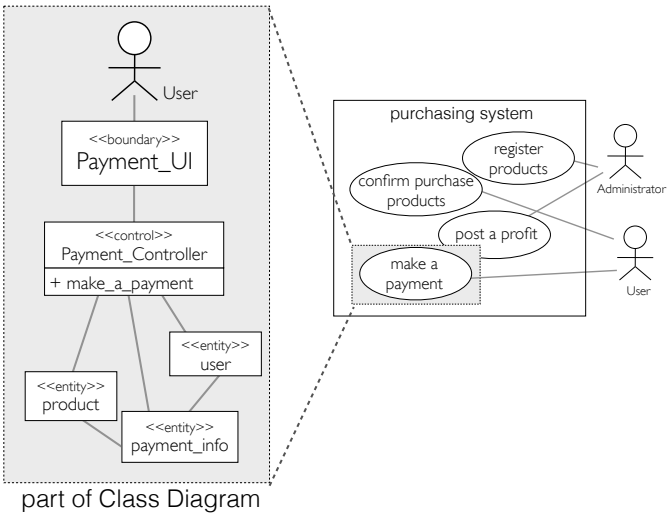

Figure 4. "Make a payment" portion of a class diagram for payment processing

For example, even if a developer intends to apply an RBAC to the model in Fig. 4, the functionality of the access control cannot be determined until the design model is tested. Thus, the measure may not mitigate or resolve the threats and vulnerabilities. Arnon et al. have suggested using a stereotype for a database application to validate security patterns [10]. Although this method will validate pattern applications structurally, it will not validate whether the pattern behavior in the model resolves vulnerabilities to threats. Figure 5 shows the conventional process for pattern applications.

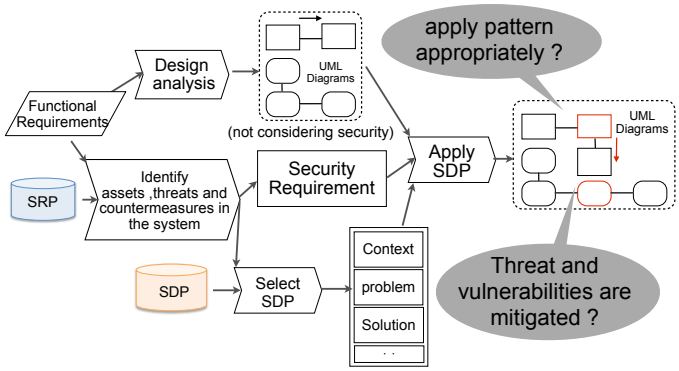

Figure 5. Conventional process

E. *Test-Driven Development (TDD)*

Test-driven development (TDD) is a software development technique that uses short development iterations based on prewritten test cases that define desired improvements or new functions. Here we use TDD for our testing process. TDD requires that developers generate automated unit tests to define code requirements prior to writing the actual code [12]. The test case represents a requirement that the program must satisfy [13].

Our method employs USE [14], which is a tool in the UML-based simulation environment that runs tests to specify and validate information systems based on a subset of the UML and the Object Constraint Language (OCL) [15]. OCL is a semiformal language that can be used to express constraints for a variety of software artifacts, and can specify constraints and other expressions in UML and other modeling languages. USE was initially implemented in Java at Bremen University (Germany), to evaluate OCL expressions via simulations. To check the OCL constraints, a developer can create an instance of a class in USE, and input a value as a test case. In this paper, we use OCL constraints as a requirement.

Our method initially executes tests in a design model that does not consider security in USE (Test First). Then our method detects vulnerabilities to threats identified in the requirement stage. Next, security patterns are applied, and the tests are re-executed to confirm that the vulnerabilities are resolved.

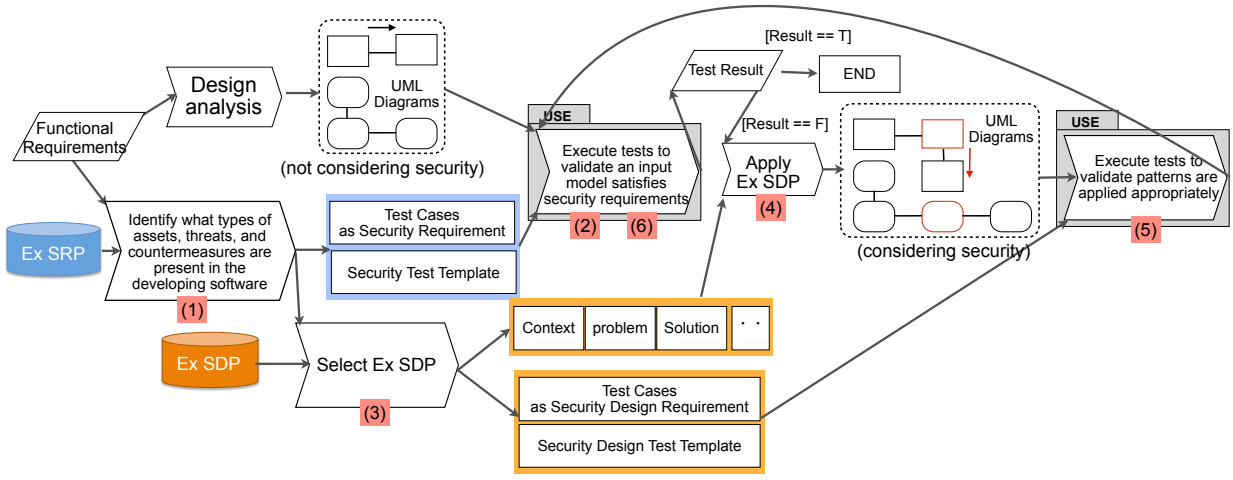

Figure 6. Process of our method

III. Our Validation Method

A. Process of Our Validation Method

Figure 6 shows the process of our method. We prepare extended SRP (Ex-SRP) and extended SDP (Ex-SDP) beforehand. These new SRP and SDP are expanded from existing SRP and SDP. A developer can execute tests and validations using these new SRP and SDP. Figure 7 shows the overall structure of Ex-SRP and Ex-SDP.

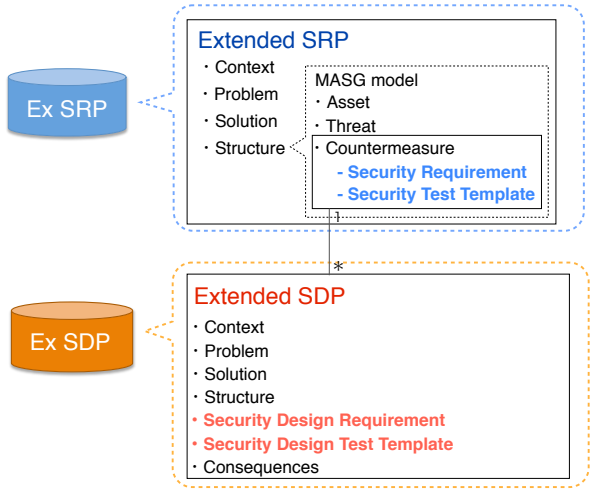

Figure 7. Overall structure of Ex-SRP and Ex-SDP

Below we briefly describe Ex-SRP and Ex-SDP, while section III B provides a concrete example.

Ex-SRP

- **Security Requirement:** Each "countermeasure" must satisfy the requirement. If a model does not satisfy the Security Requirement, then the measures do not remove vulnerabilities, and threats may exist in the system. Herein we assume that there are nine types of countermeasures: "Input and Data Validation", "Identification and Authentication", "Authorization", "Configuration Management", "Sensitive Data", "Session Management", "Cryptography", "Exception Management", and "Auditing and Logging". Each Security Requirement is prepared beforehand, assuming that these countermeasures can be referenced in the Security Frame Category [16] [17], which is Microsoft's systematic categorization of threats and vulnerabilities. In TDD, these requirements represent test cases that must be satisfied.
- **Security Test Template:** This template executes tests to validate whether a design model satisfies the Security Requirement related to each countermeasure.

Ex-SDP

- **Category:** Each Ex-SDP belongs to a unique category. For example, the "Password Design and Use" pattern belongs to "I&A", while the "RBAC" pattern belongs to "Authorization".
- **Context:** In addition to the existing "Context", we add the structure and behavior to a potential "Problem", which occurs when a situation does not satisfy the Security Requirement.
- **Structure:** The structure and behavior must constantly satisfy the Security Requirement related to the category.
- **Security Design Requirement:** To meet a requirement (constraint), the structure should be satisfied when a pattern is applied. If a model does not satisfy the Security Design Requirement, then the pattern is applied inappropriately.
- **Security Design Test Template:** This template executes tests to determine whether the design model satisfies the Security Design Requirement.

Our method involves six steps (Fig. 6).

1. In consideration of the functional requirements, Ex-SRPs are used to identify the types of assets, threats, and countermeasures present in the developing software. Additionally, Ex-SRPs determine their associations at the requirement level. Then test cases are set as Security Requirements for the target process based on the countermeasures.
2. As an input model, a design class diagram, which does not consider security, is used to execute tests on USE via the Security Test Template that prepared each Ex-SRP "countermeasure". At this point, it must be confirmed that the input model does not satisfy the Security Requirement; that is, the vulnerabilities to the threat identified at the requirement stage can be detected.
3. Ex-SDPs related to the "countermeasures" of Ex-SRP are selected.
4. Then Ex-SDPs are applied. Specifically, the "Structure" of Ex-SDPs is applied to the input model that does not consider security.
5. Tests are executed in USE using the Security Design Test Template to validate the appropriateness of each Ex-SDP; that is, whether the Security Design Requirements are satisfied is confirmed.
6. Finally, tests are re-executed using the Security Test Template to validate whether the applied patterns satisfy the Security Requirement; that is, whether the vulnerabilities to the threats identified at the requirement stage are resolved is determined. If the results of security test returns true, the process is complete.

B. Examples of Ex-SRP and Ex-SDP

In addition to explaining Ex-SRP and Ex-SDP concretely, we describe how the model uses these extended patterns to satisfy the Security Requirement and the Security Design Requirement. Expansion details are described as examples of

the "I&A" and "Password Design and Use" countermeasures for Ex-SRP and Ex-SDP, respectively.

1) *I&A*

TABLE I. Security Requirement of I&A

		1	2
Conditions	regular user	Yes	No
Actions	execute process that requires I&A	×	
	not execute process that requires I&A		×

```
context subject_controller
 inv Security Requirement :
  if self.UI.actor.regular_user = true then
   self.subject_function = true
 else
   self.subject_function = false
 endif
```

Figure 8. Security Requirement of I&A (OCL)

Table I and Fig. 8 show the Security Requirement of I&A, which is only actors who are regular users can execute processes that require I&A. Table I is in list form, while Fig. 8 is an OCL statement of this concept. Figure 9 shows an example of the Security Test Template for model testing. In USE, a developer can create an instance of a class, and input a value as a test case. This template allows input models, which do not consider security, to be tested and the OCL statement in Fig. 8 to be evaluated.

```
----------Create instances
!create Actor_1 : Actor
!create UI : UI
!create Subject_function : Subject_function
!create entity_1 : entity
        .
        .
---------------------------

--------- Insert associations
!insert (Actor_1, UI) into assignedTo
        .
        .
---------------------------

--------- Set Test Case
!set Actor_1.name := 'XXXX'
!set Actor_1.regular_user := true
!set entity_1.attribute := y
        .
        .
---------------------------

--------- Execute Method
!openter Subject_function subject_function()
```

part of security test template

Figure 9. Example of a Security Test Template

In the I&A security test, two conditions ("regular user" and "non-regular user") are used to validate whether an actor can execute a process. If a model does not satisfy this Security Requirement, then the I&A measure for the vulnerabilities is improperly considered, and the system may be vulnerable to threats.

2) *Password Design and Use pattern*

Figure 10 shows the structure and behavior for a potential "Problem". Although "subject_function" of "Subject_Controller" is a required I&A function, illegal situations where a non-regular user can access assets without the certification process exist. Thus, the structure and behavior in Fig. 10 do not satisfy the Security Requirement in Table I and Fig. 8.

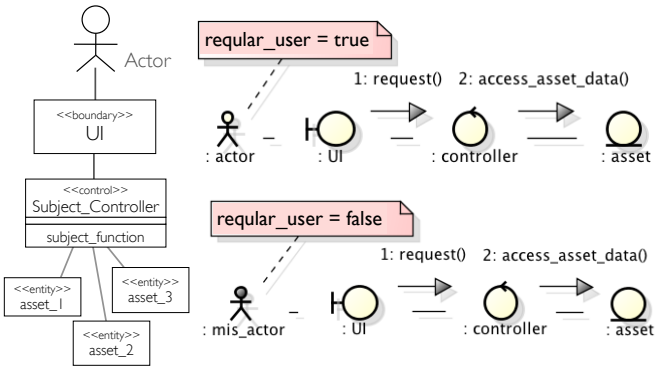

Figure 10. Structure and behavior (not satisfying the I&A Security Requirement)

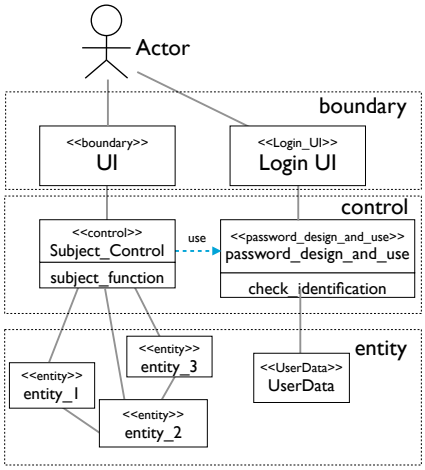

Figure 11. Structure of Password Design and Use

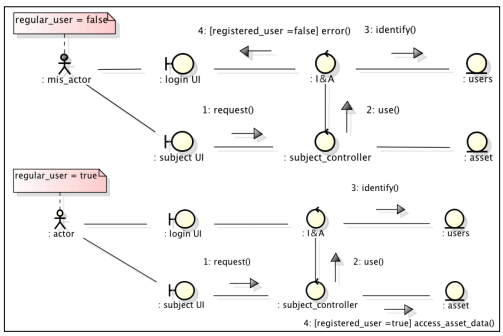

Figure 12. Behavior of Password Design and Use

Figures 11 and 12 satisfy the Security Requirement in Table I and Fig. 8. In the Password Design and Use pattern, stereotypes, such as *<<Login_UI>>*, *<<Password_Design_And_Use>>*, and *<<UserData>>*, are elements of the pattern. Thus, "Subject Controller" employs *<<Password_Design_And_Use>>*, which is part of the I&A function. In this scenario, if an actor is a non-regular user, USE outputs an error, and the actor is unable to execute processes requiring I&A. This security capability is realized because the model applies patterns that satisfy the Security Design Requirement (SDR).

TABLE II. SDR of Password Design and Use pattern

		1	2
Conditions	the same ID and Password that are inputted into "Login_UI" exist in "User_Data",	Yes	No
Actions	consider regular user	×	
	consider non-regular user		×
	execute process that requires I&A	×	
	not execute process that requires I&A		×

```
context subject_controller
 inv check_id_and_pass:
  if self.password_design_and_use.User_Data->exists(p |
       p.id = self. password_design_and_use.Login_UI.id and
       p.pass = self. password_design_and_use.Login_UI.pass)
  then
     self.UI.actor.regular_user = true  and self.subject_function = true
  else
     self.UI.actor.regular_user = false  and self.subject_function = false
  endif
```

Figure 13. SDR of Password Design and Use pattern (OCL)

Table II and Fig. 13 show the Security Design Requirement of the Password Design and Use pattern in list and OCL statement form, respectively., The Password Design and Use pattern stipulates that the ID and password inputted in *<<Login_UI>>* must exist in *<<User_Data>>* in order for an actor to be considered a regular user, who can execute processes requiring I&A. This condition of the Security Design Requirement satisfies the Security Requirement of I&A in Table I and Fig. 8. In other words, if patterns are applied appropriately, then the output model will simultaneously satisfy the Security Design Requirement and the Security Requirement.

Figure 14 shows the Security Design Test Template, which was prepared in USE, used to evaluate whether the output model satisfies the Security Design Requirement shown in Fig. 13. Using an OCL statement to describe the Security Requirement and the Security Design Requirement can simultaneously validate both requirements. The former confirms the pattern is appropriately applied, while the latter determines the presence of vulnerabilities.

```
----------Create instances
!create Actor_1 : Actor
!create UI : UI
!create Subject_function : Subject_function

!create Login_UI : Login_UI
!create password_design_and_use : password_design_and_use
!create User_Data : User_Data
   .
   .
--------- Insert associations
!insert (Actor_1, UI) into assignedTo
   .
   .
--------- Set Test Case
!set Actor_1.name := XXXX

!set Login_UI.id := z
!set Login_UI.pass := 'xxxxxx'

!set User_Data.id := z
!set User_Data.pass := 'xxxxxx'
!set User_Data.name := 'XXXX'
      .
      .
--------- Execute Method
!openter password_design_and_use check_identification()
!openter Subject_function subject_function()
```

part of security design test template

Figure 14. Example of the Security Design Test Template

C. *Example of the Validation Process*

Here we apply our method to a purchasing system on the Web as an example validation process. We initially identified and modeled the assets, threats, and countermeasures in the system by referring to the Ex-SRPs of the requirement called "the commercial transaction on the Web". Next we executed tests of the input model in USE to validate whether vulnerabilities to threats identified by Ex-SRPs are detected. After confirming that these vulnerabilities really exist in the input model, we applied Ex-SDPs. Finally, we re-executed the tests to confirm that the vulnerabilities are resolved due to an appropriate pattern application.

Step 1: The example validation process assumed that the MASG model in Fig. 1 is used and the task of security measure for the "make a payment" process in Fig. 4 is performed. "I&A", "Input Data and Validation", and "Access Control" are countermeasures for "Spoofing", "Privilege Exploitation", and "SQL Injection" in the "make a payment" process, respectively. Then by referencing the Security Requirement used for each Ex-SRP countermeasure, the set for the Security Requirement should be satisfied in the "make a payment" process. Table III and Fig. 15 show the Security Requirement for the "make a payment" process.

For the "make a payment" process, valid data must be inputted for a regular user to have permission to execute the "make a payment process", which is a combination of multiple Security Requirements: "I&A", "Input Data and Validation", and "Access Control". These requirements represent test cases in the TDD process.

TABLE III. Security Requirements for the "make a payment" process

		1	2	3	4	5	6	7	8
Conditions	regular user	Yes	Yes	Yes	Yes	No	No	No	No
	have access permission	Yes	Yes	No	No	Yes	Yes	No	No
	use valid input data	Yes	No	Yes	No	Yes	No	Yes	No
Actions	execute "make a payment" process	×							
	not execute "make a payment" process		×	×	×	×	×	×	×

```
context payment_controller
 inv SecurityRequirement :
  if self.payment_UI.User.regular_user = true and
     self.payment_UI.User.right = true and
     self.payment_UI.valid_input_data = true then
   self.make_a_payment = true
 else
   self.make_a_payment = false
 endif
```

Figure 15. Security Requirements for the "make a payment" process (OCL)

Step 2: We executed a model test on USE using the Security Test Template to determine whether the input model that does not consider security satisfies the Security Requirements in Fig. 15. If the Security Requirement is not satisfied, then the appropriate countermeasures are not taken, and the threats identified using Ex-SRP are possible. Table IV shows the results.

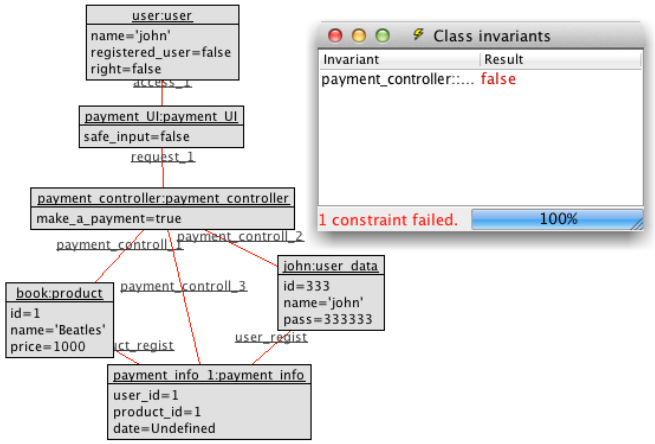

Figure 16. Conditions of the Security Test in USE

TABLE IV. Results of the Security Test

		1	2	3	4	5	6	7	8
Conditions	regular user	Yes	Yes	Yes	Yes	No	No	No	No
	have access permission	Yes	Yes	No	No	Yes	Yes	No	No
	use valid input data	Yes	No	Yes	No	Yes	No	Yes	No
Actions	execute "make a payment" process	×	×	×	×	×	×	×	×
	not execute "make a payment" process								

Figure 16 shows a case where "regular user", "have access permission", and "use valid input data" are all false (test case 8, Table IV). Because the input model lacks object constraints, an actor may carry out "make_a_payment = true"; that is, an actor can execute the "make a payment" process without being a regular user or permission. Hence, the input model not considering security does not satisfy the Security Requirement of the "make a payment" process, and the evaluation of OCL on USE becomes "false" in Fig. 16. Table IV shows the results of the eight test cases where only case 1 satisfies the Security Requirements in Table III and Fig. 15. In this way, countermeasures "I&A", "Input Data and Validation", and "Access Control" are confirmed necessary.

Step 3: We selected Ex-SDP related to the countermeasures of Ex-SRP to add the structure and behavior with security capabilities. In this example, we used the Ex-SDPs shown in Fig. 17.

Countermeasure	Ex SDP
I&A	Password Design and Use
Access Control	RBAC (Role-Based Access Control)
Input and Data Validation	Prevent SQL Injection

Figure 17. Selected Ex-SDPs

Step 4: We apply these Ex-SDPs, i.e. "Password Design and Use", "RBAC", and "Prevent SQL Injection". Figure 18 shows the structure after applying the pattern to an input model. During pattern application, we bind the pattern elements with a stereotype similar to that shown in Fig. 18.

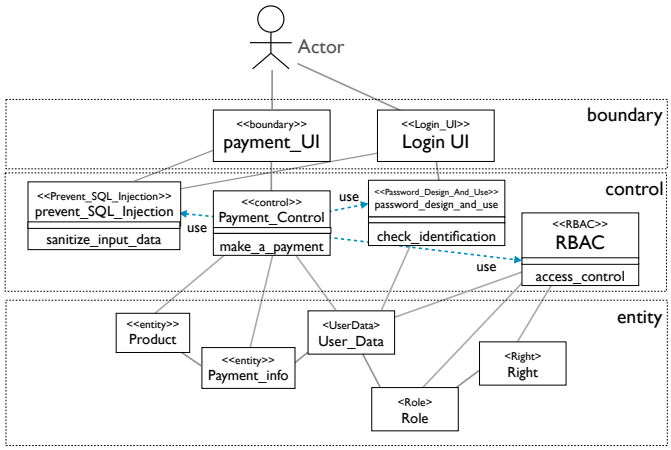

Figure 18. Structure used to apply patterns

Step 5: To validate whether patterns are applied appropriately to the "make a payment" process, the Security Design Requirement of the "make a payment" process must be validated. Table V and Fig. 19 show the Security Design Requirements of the "make a payment" process.

TABLE V. Security Design Requirements for the “make a payment” process

		1	2	3	4	5	6	7	8
Conditions	the same ID and Password that are inputted into “Login_UI” exist in "User_Data”,	Yes	Yes	Yes	Yes	No	No	No	No
	access permission is given in “Role” to which an actor belongs	Yes	Yes	No	No	Yes	Yes	No	No
	sanitize input data in UI	Yes	No	Yes	No	Yes	No	Yes	No
Actions	consider regular user	×	×	×	×				
	consider non-regular user					×	×	×	×
	considers that an actor have access permission	×	×			×	×		
	consider that an actor does not have access permission			×	×			×	×
	consider that valid input data is used	×		×		×		×	
	consider that invalid input data is used		×		×		×		×
	execute “make a payment” process	×							
	not execute “make a payment” process		×	×	×	×	×	×	×

```
context payment_controller
  inv check_id_and_pass:
    if self.password_design_and_use.User_Data->exists(p |
                p.id = self.password_design_and_use.Login_UI.id and
                p.pass = self.password_design_and_use.Login_UI.pass)
  then
      self.Payment_UI.actor.regular_user = true
  else
      self.Payment_UI.actor.regular_user = false
  endif

context payment_controller
  inv access_control:
    if self.RBAC.Right->exists(p |
          p.right = true and
          p.role_id = p.Role.id and
          p.role_id = p.Role.User_Data.role_id )
  then
      self.Payment_UI.actor.right = true
  else
      self.Payment_UI.actor.right = false
  endif

context payment_controller
  inv sanitize_input_data_payment_UI:
    if self.Payment_UI.Prevent_SQL_Injection.sanitize_input_data = true
  then
      self.Payment_UI.valid_input_data = true
  else
      self.Payment_UI.valid_input_data = false
  endif

context payment_controller
  inv sanitize_input_data_login_UI:
    if self.password_design_and_use.Login_UI.Prevent_SQL_Injection.sanitize_input_data = true
  then
      self.password_design_and_use.Login_UI.valid_input_data = true
  else
      self.password_design_and_use.Login_UI.valid_input_data = false
  endif

context payment_controller
  inv security design requirement:
    if self.Payment_UI.actor.regular_user = true and
       self.Payment_UI.actor.right = true and
       self.Payment_UI.valid_input_data = true and
       self.password_design_and_use.Login_UI.valid_input_data = true
  then
      self.make_a_payment = true
  else
      self.make_a_payment = false
  endif
```

Figure 19. Security Design Requirements of “make a payment” (OCL)

To validate whether the model shown in Fig. 18 satisfies the Security Design Requirements in Fig. 19, we executed model tests in USE using the Security Design Test Template. Figure 20 shows the conditions of the Security Design Test in USE.

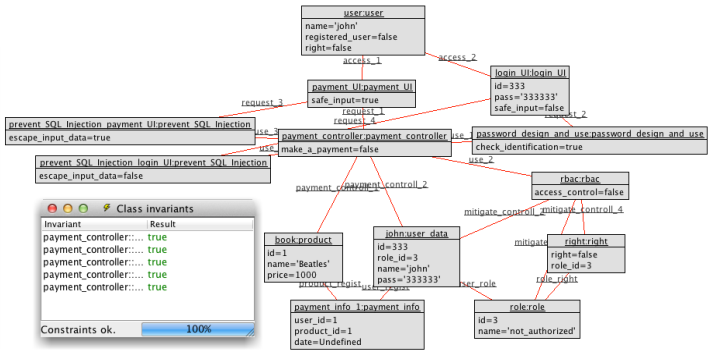

Figure 20. Conditions of the Security Design Test in USE

Figure 20 shows a case where the inputted ID and Password into *<<Login_UI>>* exists in *<<User_Data>>*, but access permission is not given for the “Role” of the actor and the system does not sanitize the “UI” input data (case 4, Table V). Prior to applying patterns, an actor can execute the “make a payment” process, even if the actor does not have permission or inputs invalid data because USE outputs “make_a_payment = true”. After patterns are applied, USE outputs "make a payment = false" and the actor cannot execute the “make_a_payment” process because access permission is not specified in “Role” and the system assumes invalid data is used in “UI”. By executing all the test cases, we confirm that the output model after a pattern application satisfies the Security Design Requirements for the “make a payment” process.

Step 6: Finally we re-executed the Security Test to validate that the output model with a pattern application satisfies both the Security Design Requirement and the Security Requirement. If it satisfies the Security Requirement, then the countermeasures appropriately resolve vulnerabilities in the “make a payment” process.

To summarize, we applied Ex-SDPs for the “make_a_payment” process that required “I&A”, “Input Data and Validation”, and “Access Control”, and executed a model test in USE. The Security Test confirmed that the initial input model did not satisfy the Security Requirement of the “make a payment” process. Then the Security Design Test evaluated whether the output model applied patterns to satisfy the Security Design Requirement of the “make a payment” process. Finally, the Security Test was re-executed to verify that the revised model applied patterns to satisfy the Security Requirement. In this manner, the appropriate application of security design patterns and the existence of vulnerabilities to threats identified at a requirements stage before and after pattern application could be validated.

D. *Limitations*

Our method has a few limitations. Because tests are executed based on threats and countermeasures identified in the requirement stage, the presence of threats not identified in the

requirement stage cannot be verified. In addition, the criterion for Ex-SDP, which belongs to the countermeasures (e.g., to realize the I&A function, we considered which patterns to employ, such as “Password Design and Use”, “Biometrics”, or “PKI”), may be out of range because the range is influenced by the security policy, platform, and risk analysis.

IV. Case Study and Discussion

Here we apply our method to a check-in system for a hotel as a case study and to evaluate RQ1 and RQ2. This system is treated with the NII/TopSE project "aspect oriented lecture" [18]. After referencing the Ex-SRP, we identified the threats to the check-in process (“Spoofing” and “SQL Injection”) and assumed that “I&A” and “Input and Data Validation” are effective countermeasures. In other words, when an actor check-ins from the check-in screen, the system does not have a function to determine whether an actor previously reserved a room or if the inputted data is valid. Figure 21 shows the design class diagram that does not consider security as the input model, while Fig. 22 shows the Security Requirements that should be satisfied during the check-in process, which are a combination of “I&A” and “Input Data and Validation”.

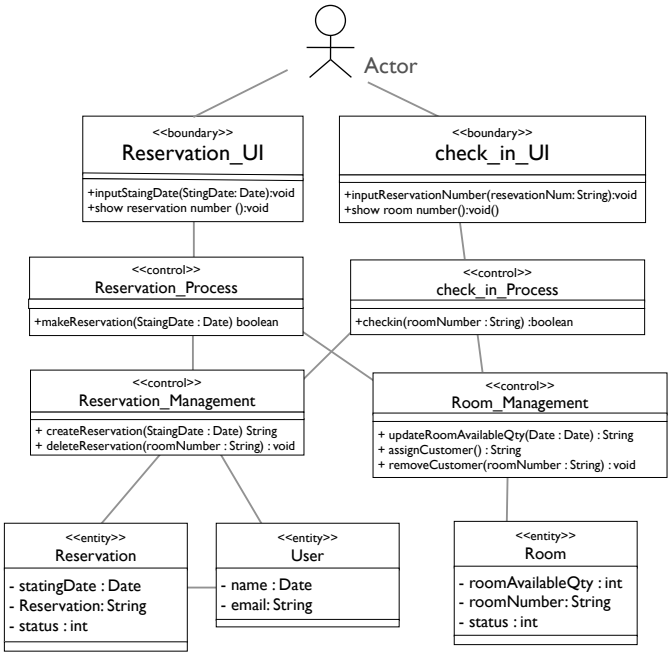

Figure 21. Class Diagram for a hotel check-in process (not considering security)

```
context check_in_Process
 inv SecurityRequirement :
  if self.payment_UI.User.regular_user = true and
     self.payment_UI.valid_input_data = true then
   self.chckIn = true
 else
   self.chckIn = false
 endif
```

Figure 22. Security Requirement of for the check-in process (OCL)

We then validated whether the input model satisfies the Security Requirements of the check-in function and whether the vulnerabilities are resolved upon applying patterns in USE. After confirming that the input model does not satisfy the Security Requirement, we applied "Password Design and Use" and "Prevent SQL Injection" as Ex-SDPs. Figure 23 shows the structure of the applied patterns in this input model.

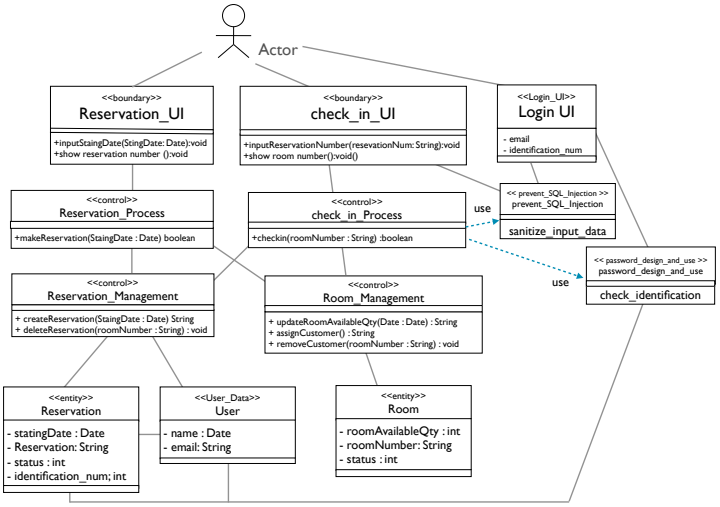

Figure 23. Structure that applied patterns

The applied patterns in this model realize an “I&A” function, allowing the actor to input *<<email>>* and *<<identification_num>>* in *<<Login_UI>>*. In addition, this model realizes an "Input Data and Validation" function via a sanitizing process, *<<Check_in_UI>>* and *<<Login_UI>>*. To confirm whether the structure and behavior of the applied patterns operate appropriately, we validated the Security Design Requirement of the check-in process using model tests. Table VI shows the Security Design Requirements of the check-in process.

TABLE VI. Security Design Requirements of the check-in process

		1	2	3	4
Conditions	“email” and “identification_num” that are inputed into “Login UI” exist in “User Data” and “Reservation”	Yes	Yes	No	No
	sanitize input data in UI	Yes	No	Yes	No
Actions	consider regular user	×	×		
	consider non-regular user			×	×
	consider that valid input data is used	×		×	
	consider that invalid input data is used		×		×
	execute “check-in” process	×			
	not execute “check-in” process		×	×	×

We executed model tests (Security Design Test) for the four test cases in Table VI. The model after the pattern application satisfies the Security Design Requirement in Fig. 22, validating the appropriateness of the pattern application. Thus, the proposed method answers **RQ1**.

Finally we validated whether the model after pattern application satisfies the Security Requirements of the check-in process via model testing (Security Test). The retest confirmed that the application is successful because the Security Design Requirement and the Security Requirement are simultaneously satisfied. Consequently, the Security Test validated the existence of vulnerabilities identified in the requirement stage before and after pattern application. Thus, the proposed method answers **RQ2**.

V. THREATS TO VALIDITY

A. *Threats to internal validity*

Herein patterns were prepared and then subsequently applied to the model. Although a test template may eliminate human dependency, the effectiveness of the template should be confirmed when employed by a developer unfamiliar with our method.

B. *Threats to external validity*

We did not verify whether our method is applicable to any type of model. Although we used representative patterns and a typical model for software development, we should confirm that our method applies patterns to more general examples.

VI. CONCLUSION AND FUTURE WORK

If a software developer is not a security expert, patterns may be inappropriately applied. Even if patterns are properly applied, threats and vulnerabilities may not be mitigated or resolved. Herein we propose a validation method for a security design pattern using a model test in the UML model simulation environment. Specifically, assets, threats, and countermeasures are identified in the target system during an early stage of development. We validated the appropriate application of the pattern and the existence of vulnerabilities that are identified in the first stage of the design model.

This method offers two significant contributions. First, Ex-SRP and Ex-SDP, which are new extended security patterns that include both requirement- and design-level patterns, are combined to realize validation via model tests. Second, a new model testing process using these extended patterns is proposed. In the future, we intend to develop a test execution that is independent of the USE model description language. Although we prepared the test execution templates to handle USE, automatic transformation from a model would realize a smoother test.

REFERENCES

[1] N.Yoshioka, H.Washizaki, K.Maruyama “A Survey on Security Patterns”, Progress in Informatics, No.5, pp. 35-47. 2008

[2] M.Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, Peter Sommerlad. “SECURITY PATTERNS”. Wikey. 2006

[3] T. Heyman, K. Yskout, R. Scandariato, and W. Joosen, “An analysis of the security patterns landscape,” in Proceedings of the Third International Workshop on Software Engineering for Secure Systems, ser. SESS ’07. IEEE Computer Society, 2007, pp. 3–.

[4] OMG, Unified Modeling Language (UML). http://www.uml.org/

[5] Jan Jurijens “Secure Systems Development with UML” 2004,Springer.

[6] Y.Torsten Lodderstedt David A. Basin Jürgen Doser “SecureUML: A UML-Based Modeling Language for Model-Driven Security” 2002

[7] T.Okubo, H.Kaiya, N.Yoshikawa Effective Security Impact Analysis with Patterns for SoftwareEnhancement IJSSE 3(1): 37-61 (2012)

[8] T.Okubo, K.taguch, N.Yoshioka Misuse Cases + Assets + Security Goals International Conference on Computational Science and Engineering 2009.

[9] Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements by misuse cases. In TOOLS (37), pages 120–131. IEEE Computer Society, 2000.

[10] T. Peng, J. Dong, and Y. Zhao, “Verifying Behavioral Correctness of Design Pattern Implementation,” Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE'2008)

[11] Arnon Sturm, Jenny Abramov, Peretz Shoval Validating and Implementing Security Patterns for Database Applications SPAQu ’09 2009

[12] Heejin Kim, Byoungju Choi, Seokjin Yoon “Performance testing based on test-driven development for mobile applications” ICUIMC '09:

[13] Steven Fraser, Dave Astels, Kent Beck, Barry Boehm, John McGregor, James Newkirk, Charlie Poole”Discipline and practices of TDD (test driven development)” OOPSLA '03:

[14] USE http://www.db.informatik.uni-bremen.de/projects/USE/index.html

[15] Jos. Warmer, Annke Kleppe "The Object Constraint Language - Precise Modeling with UML" Addison-Wesley, 1999

[16] M. Howard and S. Lipner, The Security Development Lifecy-cle. Microsoft, 2006.

[17] Web Application Security Frame http://msdn.microsoft.com/en-us/library/ms978518

[18] NII/TopSE project "aspect oriented lecture" http://topse.jp