An Evaluation of a Use Case Driven Requirements Analysis Using Web UI Prototype Generation Tool

SHINPEI OGATA
Function Control System,
Graduate School of Engineering
Shibaura Institute of Technology
307 Fukasaku, Minuma-ku Saitama-City,
Saitama 337-8570
JAPAN
m709101@sic.shibaura-it.ac.jp
http://www.sayo.se.shibaura-it.ac.jp/

SAEKO MATSUURA
Department of Electronic Information Systems,
College of System Engineering and Science
Shibaura Institute of Technology
307 Fukasaku, Minuma-ku Saitama-City,
Saitama 337-8570
JAPAN
matsuura@se.shibaura-it.ac.jp
http://www.sayo.se.shibaura-it.ac.jp/

Abstract: It has been widely acknowledged that a major reason for software project failure is the ambiguous and incomplete customer's requirements. One of their main cautions is the inconsistency between implementation image and specification at an early stage of software development. Therefore, we have been proposed a method to automatically generate a user interface prototype from a requirements analysis model in Unified Modeling Language in order to decrease inconsistency between them for business Web application development. In this paper, we have compared our proposal with traditional use case modeling to evaluate the effectiveness of our proposal.

Key-Words: Use case, Prototyping, Web application, Requirements analysis, Unified Modeling Language

1 Introduction
It has been widely acknowledged that a major reason for software project failure is the ambiguous and incomplete customer's requirements. Furthermore, data structure and flows are quite complex in business system development. As the customers are generally familiar with such a complex business rules, they can evaluate the adequacy of the requirements specification, so that the ambiguity and incompleteness would be reduced.

A use case driven requirement analysis is a typical object-oriented analysis method of specifying how a system should interact with the users and other external systems. A use case model generally consists of a use case diagram in Unified Modeling Language (UML) [1] and several use case templates [2, 4].

There are two advantages of use case driven requirements analysis. Firstly, the use case templates are written in natural language and need avoiding the use of technical terms about implementation, so that the customers can understand them easily. Secondly, a use case model is required to focus on modeling the interaction so that it contains a lot of core requirements which are the basis of the requirements of user interface (UI), performance, data format and business rules. Therefore, many researchers accept the concept of use case as effective one in several such fields in requirements engineering as requirements elicitation, analysis, validation and verification [5-8, 13, 14].

If UML model precisely defines the customers' requirements, it can be handled as requirements specification. However, there are three problems to define the interaction as the requirements precisely.

Firstly, though technical-independent terms are familiar to the customers, the operation image of a system is not realistic for them. For example, it is difficult for them to understand the input means and concrete data of input/output data. Secondly, the description in natural language makes the developers and customers imagine implementation image arbitrarily. Therefore, natural language is apt to cause misunderstanding the requirements between the developers and the customers. Thirdly, the classes, generally, is derived from the terms of use case templates in natural language. However, it is difficult for the developers to correctly derive the classes because these terms represent various instances of the potential classes.

We have proposed a method to automatically generate UI prototype from an original requirements analysis model (RA model) in UML for business Web application development so that the developers can resolve above-mentioned problems [9, 10]. The RA model consists of activity diagrams, a class diagram...
and object diagrams. The UI prototype is a tool to understand a system image easily and implements specific aspect of a system such as only UI [11].

In our method of automatic prototype generation, the first and second problems are resolved by clarifying the correspondence between the RA model and implementation image as UI prototype. Furthermore, the third problem is resolved by precisely collaborating among a classifier of object nodes in the activity diagram, a class in the class diagram and a classifier of instance specifications in the object diagram.

In this paper, we evaluate the effectiveness of a requirements analysis method applying our automatic prototype generation method compared with a traditional use case modeling. In the evaluation, we measured the time taken by each method and the consistency between the RA model and implementation image.

2 Traditional Use Case Modeling
According to a standard of software requirements specifications [3], the functions include the follows: validity checks on the inputs; exact sequence of operations; responses to abnormal situations and etc. These items become clear in the interaction between actors and a system. To capture the functional requirements correctly, use case is proposed [2]. A use case is the behavior of a system from the actor's viewpoint. An actor is a role to the system and is classified into an authority or an external system.

2.1 Use Case Model
A use case model consists of a use case diagram in UML and use case templates every use case.

Definition of a use case diagram:
A use case diagram expresses the principal system's behavior and consists of use cases, actors and relationships between them.

Definition of use case templates:
A use case template represents details of the interaction between actors and a system. It consists of title, actors, basic flow, alternative flows, exceptional flows, preconditions and postconditions. The basic flow and alternative flows (normal flows) are the flows to accomplish workflow correctly. Exceptional flows are the flow to recover from an error state. These flows are represented as sequence of steps. A step is numbered and ordered action needed the interaction.

Definition of a class diagram:
At an early stage of a system development, the developers define a class diagram of concept level in UML as a set of data structure. In the traditional use case modeling, classes are elicited based on the data expressions which are appeared in a use case template.

2.2 Limitations of Use Case Modeling
There are three difficult points to define sufficiently and unambiguously the functional requirements into the use case model as follows.
Firstly, use case templates which focus on ease of read for the customers are often ambiguous by too simple. We consider the points which the customers must understand precisely are follows:
(1) What workflows can each use case accomplish?
(2) What sequence of operation steps can the users execute the use cases by?
(3) What flows of alternative or exceptional can the users take by their various inputs which include correct input or incorrect input?
(4) At each step, what data can the users input or confirm? The data includes the name, the values, the format of values and the data structure.
(5) What input means can the users input data with? For example, in a Web page, input means are a text input, check boxes, radio buttons and etc.

These five items are needed inevitably to ensure the customers the workflows which the users can accomplish correctly. A use case template has weakness to (4) and (5) because the template avoids implementation-dependent description and does not represent concrete data. However, if the templates define the details of their items, their contents become more complex and become unreadable.

Secondly, the gap between the customers and the developers is strongly caused by using specifications which are described only in natural language on discussing and understanding the requirements. As a cause of the gap, each customer and each developer imagine the arbitrary implementation image from the specification in natural language.

Thirdly, whether developers can ensure the consistency between a use case model and a class diagram depends on their ability. Generally, classes are derived from data expressions in use case templates. However, it is difficult for the developers to elicit the potential class because the terms in use case templates are various instances of the potential classes. As a result, it is not easy to sufficiently and correctly derive the classes from the terms.
3 Use Case Modeling with Automatic Prototype Generation

3.1 Modeling Cycle in Requirements Analysis

Step 1. Definition of the RA model:
Developers define a RA model focused on the functional requirements using JUDE [12] which is a UML modeling tool. The RA model includes activity diagrams, a class diagram and object diagrams. The JUDE can associate precisely among a class, the classifier of object nodes and the classifier of instance specifications. The output of this process is an instance of the RA model.

Step 2. Generation of a UI prototype:
A UI prototype which is generated from the RA model is one of the implementation views captured by integrated among all diagrams in the RA model. The output of this process is a set of pages in Hyper Text Markup Language (HTML) which have no functions.

Step 3. Validation of the RA model through the UI prototype:
The developers and the customers validate the RA model about five items described in section 2.2 using the generated prototype. The output of this process is problems which includes definition mistakes and change requirements discovered on the validation.

Step 4. Refinement of the RA model:
The developers modify the RA model by correcting the problems discovered in the validation at step 3. The output of this process is the refined RA model. The developers conduct from step 2 to step 4 iteratively until the customers satisfy the generated UI prototype which implies the RA model.

3.2 Definition and Validation of the RA Model

Fig. 1 shows the overview of definition process of the RA model. As input to start the process, requirements provided by the customers, initial use cases and actors analyzed by the developers are expected.

A strong characteristic of the proposed method is that a UI prototype can be automatically generated at each stage of defining each kind of diagram so that developers can clearly and easily understand the correspondence between the contents of a UI prototype and the contents of each kind of diagram.

Definition of Interaction Activity Diagrams:
An interaction activity diagram defines interactions between actors and a system using activity diagram in UML with the same concept of the use case template.

Fig. 2 Interaction Activity Diagram of "Add New Student" Service of a Class Management System

The sequence of operations where the users can execute the services by is represented as sequence of actions which is conducted by the actors and system shown as each partition. In Fig. 2, the "Administrator" partition represents the authority. Other partitions represent roles of the system. The flows of alternative or exceptional which the users can take are represented as conditional branches using decision nodes and guard conditions. The data which the users can input or confirm is object nodes which flow toward the partition of an authority. For example, "studentList" object node corresponds with the visible data for the users. The input means where the users can input data with is represented as the verb of the actions in the partition of an authority. For example, "input grade" action converts into text format on a UI prototype generated from this model.

Definition of a Class Diagram:
A class diagram defines data structures as classes of concept level shown as Fig. 3. The visible data for the
users is important elements to accomplish the business workflow correctly. Therefore, classes are defined as the structure of object nodes in the interaction activity diagram. These classes categorize into boundary candidate and entity candidate.

![Fig.3 Class Diagram for "Add a New Student" Service](image)

**Definition of Object Diagrams:**
Concrete data is the easiest information to understand for the customers. Thus, concrete data should be used to validate the requirements effectively. The object diagram defines instance specifications as concrete data under a specific usage situation of a system.

![Fig.4 A Page of Generated UI Prototype](image)

Fig.4 represents a page of UI prototype generated at a stage of defining the interaction activity diagrams, the class diagram and the object diagrams. The page, input/output items and input means are derived from the interaction activity diagram. The tables as data structure are derived from the class diagram. The concrete data is derived from the object diagram.

**Definition of Scenarios:**
The developers must guarantee to the customers to correctly accomplish business workflow under a specific situation expected by the customers. To achieve this purpose, a path of traversal interaction flows is defined as a scenario in specific text format based on the interaction activity diagram. Furthermore, the scenario defines concrete data which corresponds with the object nodes appeared on the path.

**Definition of a Navigation Model:**
The services should be validated from the viewpoint of service integration finally. A navigation model defines sequence of invoking service. It is defined using activity diagram in UML. In the navigation model, an action corresponds with a service and a partition corresponds with an authority.

4 An Experiment for Evaluation of the Proposed Method

4.1 Overview of Experiment
To evaluate the effectiveness of the proposed method, each model defined by the proposed method and by traditional use case modeling (traditional method) is compared. In this experiment, there are two points of view for evaluation. One is the time taken for requirements analysis until affording satisfaction to the customer. The other is the consistency the RA model and the prototype as a quality of the RA model. The consistency is measured from whether the data name defined in the RA model completely matches the visible data name represented in the prototype. In the proposed method, this consistency is guaranteed automatically. Therefore, the inconsistency in the traditional method is measured.

The developers have basic UML knowledge and the experience of business Web application development using traditional method. The developers are described as A, B and C separately. Then, there are two target applications. One of target applications is a library management system (LMS). The other is a schedule sharing system (SSS). When the customer and developer validate the RA model, they use only prototype because the customer cannot read the model generally. Therefore, in the traditional method, the developers conduct manual prototyping. Then, the analysis is finished at the time when the prototype affords satisfaction to the customer.

4.2 Experiment Result
Table 1 represents the time taken by modeling and prototyping separately. In proposed method, though the prototyping takes no time basically, the analysis of SSS by C takes a bit of time due to the bugs of the UI prototype generation tool.

![Table 1](image)

Table 2 represents the rate of inconsistency in traditional method. Amount of pages is measured targeting all use cases. Amount of data having inconsistency is about between the implementation image and the use case templates.

![Table 2](image)

Table 3 depicts the amount of the pages which are created manually or generated automatically.

![Table 3](image)
5.1 Effectiveness of the Proposed Method

According to Table 1, the total time taken by requirements analysis with the proposed method is less than with the traditional method on the whole in spite of including the time to learn how to define the RA model in the proposed method. As the singular data, the total time taken by C-SSS is longer than by A-SSS and B-SSS. The major reason of this result was that the C defines the RA model detailed overly. The amount of pages generated automatically in Table 3 proves this fact. If each developer creates manually the prototype whose scale is 558 pages in Table 3, the time taken by manual prototyping is estimated to 88.4 hours at least by multiplying 558 pages by 9.5 minutes which are represented as the pace of one page creation in A-SSS. The estimated time is longer than the total time taken by the analysis of C-SSS. Therefore, we consider that the proposed method is effective to decrease the total time taken by analysis.

The UI prototype which contains concrete data and which represents integrated services was created whether the prototype is created automatically or not. However, in traditional method, the model defined no specification both of concrete data and flow of integrated services. In the proposed method, the concrete data is specified by the object diagram clearly. Also, the flow of integrated services is specified by the navigation model. What is worse is that the traditional method caused a great inconsistency between the UI prototype and the RA model shown as Table 2. Our proposal could avoid above-mentioned inconsistency between the prototype and the model by the automatic generation of the UI prototype from the RA model.

5.2 Problems of the Proposed Method

There are two major problems in the proposed method.

Firstly, sets of the classes which have the similar or same structure were defined.

Secondly, it is difficult for the developer to precisely specify the branch conditions which are needed at alternative flows or exceptional flows in both of the proposed method and the traditional method. For example, A defined "There is no incorrectness for all input" as branch conditions. Thus, several branch conditions cannot be understood clearly. However, if all conditions of the RA model are completely and precisely defined, the RA model becomes too complicated to read.

To resolve the first and second problems, the method to clearly and correctly define the properties of data which can accomplish the service is needed. As a policy to resolve the first problem, their properties become a baseline to derive a set of data which should be integrated. Therefore, we resolve these problems by introducing the Object Constraint Language (OCL) which is formal language for object-oriented development so that the developer can define the properties of the data clearly and precisely.

Table 1 Time Taken by Modeling and Prototyping

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtracted time [Hour]</td>
<td>19.4</td>
<td>0</td>
<td>20.6</td>
</tr>
<tr>
<td>Total time [Hour]</td>
<td>19.4</td>
<td>20.6</td>
<td>15.4</td>
</tr>
<tr>
<td>SSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subtracted time [Hour]</td>
<td>15.4</td>
<td>23.8</td>
<td>24.0</td>
</tr>
<tr>
<td>Total time [Hour]</td>
<td>39.2</td>
<td>57.4</td>
<td>87.4</td>
</tr>
</tbody>
</table>

Table 2 Rate of Inconsistency in Traditional method

<table>
<thead>
<tr>
<th></th>
<th>A-SSS</th>
<th>B-SSS</th>
<th>C-LMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)Amount of pages</td>
<td>117</td>
<td>190</td>
<td>74</td>
</tr>
<tr>
<td>(b)Amount of data in all pages</td>
<td>1202</td>
<td>1740</td>
<td>379</td>
</tr>
<tr>
<td>(c)Amount of data in the use case model</td>
<td>438</td>
<td>1018</td>
<td>189</td>
</tr>
<tr>
<td>(d)Amount of inconsistent data between pages and the model</td>
<td>764</td>
<td>722</td>
<td>190</td>
</tr>
<tr>
<td>(e)Rate of inconsistency (d) / (b) [%]</td>
<td>63.5</td>
<td>41.4</td>
<td>50.1</td>
</tr>
</tbody>
</table>

Table 3 Amount of Pages and Pace of Page Creation

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f)Amount of pages created manually</td>
<td>117 (SSS)</td>
<td>190 (SSS)</td>
<td>74 (LMS)</td>
</tr>
<tr>
<td>(g)Amount of time taken by manual creation of the pages [Hour]</td>
<td>23.8 (SSS)</td>
<td>33.4 (SSS)</td>
<td>14.1 (LMS)</td>
</tr>
<tr>
<td>(h)Pace of one page creation (g) / (f) [Minute / Page]</td>
<td>9.5 (SSS)</td>
<td>10.5 (SSS)</td>
<td>10 (LMS)</td>
</tr>
<tr>
<td>(i)Amount of pages generated automatically</td>
<td>60 (LMS)</td>
<td>119 (LMS)</td>
<td>558 (SSS)</td>
</tr>
</tbody>
</table>

Fig.5 The Example of Similar Classes

Fig.5 depicts the example of classes which can be integrated. At first, the "SearchResult" class had been the different structure from the "ScheduleOverview" class. However, the "SearchResult" class was modified to have the same structure after a certain refinement. These duplicative classes should be integrated for the latter developing process.
6 Related Work

Some [5, 6] has proposed a formal grammar of a use case template to precisely correspond between the use case model as behavioral model, and the classes as structure model. Also, this method enables to simulate use cases based on the use case model and the classes. As characteristics of the simulation, the simulation is implementation-independent and is progressed by selecting the actor's step and system's conditions to decide the path of branched flows. Jayaraman [8] proposes a simulation of use cases using a use case chart from the viewpoint of validation of the use cases. In these methods, no concrete data and no input means are represented. In our proposal, the concrete data is reflected into the generated UI prototype so that the customers can easily understand the RA model. For example, on a service of search refinement, the change of concrete data by search refinement is important to precisely understand the behavior.

Störrle [13] has proposed a method to convert activity diagrams into color petrinets in order to validation of the data flows of the activity diagrams. Also, Eshuis [14] has proposed a method to validate the workflow defined in activity diagrams. The characteristic of this method is validated from the viewpoints of event driven behavior, data, loop and real time. These methods [13, 14] focus on validation of the reachability of whether the flow can reach into expected state under specific conditions. On the other hand, our proposal focuses on validation the interaction flows from the viewpoint of what the users can input and confirm on a boundary of the system. Therefore, our proposal is uncompetitive with [13, 14] directly. There is a possibility to integrate these methods by properly combining the concrete syntax of branch conditions and properties of data.

7 Conclusion

The major contributions of our proposal through the experiment are summarized into three points. Firstly, concrete data is needed to easily understand the requirements for the customers and is clearly specified in the proposed method. Secondly, the proposed method can decrease the time taken by requirements analysis in spite of the measured time which includes learning time of the proposed method. Thirdly, although traditional method is difficult to guarantee the consistency both of data and flow between the RA model and the prototype, the proposed method can guarantee the consistency.

As the future work, we consider how to define the conditions which represent business rules using formal language such as OCL. Also, we consider how to refine the duplicative classes methodologically.

References: