# Towards an Inspection Technique for Use Case Models

Bente Anda and Dag I. K. Sjøberg

SEKE'02 -14th IEEE Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July 15-19, 2002, pp. 127-134.

---

## Abstract

A use case model describes the functional requirements of a software system and is used as input to several activities in a software development project. The quality of the use case model therefore has an important impact on the quality of the resulting software product. Software inspection is regarded as one of the most efficient methods for verifying software documents. There are inspection techniques for most documents produced in a software development project, but no comprehensive inspection technique exists for use case models. This paper presents a taxonomy of typical defects in use case models and proposes a checklist-based inspection technique for detecting such defects. This inspection technique was evaluated in two studies with undergraduate students as subjects. The results from the evaluations indicate that inspections are useful for detecting defects in use case models and motivate further studies to improve the proposed inspection technique.

**Keywords:** Use cases, Inspections

## 1 Introduction

In a use case driven software development process, a use case model is used as input to planning and estimating the software development project as well as to design and testing. A use case model may also be part of the contract between the customers and the developers regarding the functionality of a system. The quality of a use case model in terms of correct, complete, consistent and well understood functional requirements is thus important for the quality of the resulting software product.

Inspections [7] have proved to be an efficient means for detecting defects and improving quality in software documents. The structuring of the functional requirements in a use case model motivates an inspection technique with strategies for discovering defects adapted to this particular structure. The literature on use case models recommends reviews of the use case model to assure quality [3,10,16], and many organizations conduct such reviews with varying degree of formality. The increasing use of UML has motivated the development of a family of reading techniques for UML diagrams [18], but to the knowledge of the authors, no comprehensive inspection technique exists for use case models.

Several guidelines for constructing use case models exist. We conducted an experiment to evaluate the effects of two different sets of guidelines [2]. The results from that experiment show that the use of guidelines has an effect on the quality of the resulting use case models. This motivated a study to investigate how the quality of a use case model can be further improved through the use of an inspection technique.

Knowledge of typical defects is a prerequisite for developing and evaluating an inspection technique for use case models. Therefore, a taxonomy of defects in use case models and their consequences, was developed. The inspection technique is based on the taxonomy and on several recommendations for checklists found in the literature.

Any new technique should be evaluated, and the inspection technique was evaluated in a student project of a large undergraduate course in software engineering, and in a controlled experiment with 45 students as subjects.

The conducted studies indicate that inspections are useful for detecting defects in use case models, and suggest how the proposed inspection technique can be improved. Future work will focus on developing a basic technique that can be calibrated to suit a particular organization or application domain.

The remainder of this paper is organized as follows. Section 2 includes a definition of software inspections and describes different inspection techniques for requirements specifications. Section 3 presents a taxonomy of typical defects in use case models. Section 4 describes the proposed inspection technique. Section 5 reports the studies undertaken to evaluate the inspection technique. Section 6 concludes and suggests future work.

## 2 Software Inspections

This section describes the technique software inspection and the related techniques reviews and walkthroughs. Some particular inspection techniques for requirements specifications are also described.

### 2.1 Inspections, Reviews and Walkthroughs

An *inspection* is defined as a formal evaluation technique in which software requirements, design or code are examined in details by a person or group to detect defects, violations of development standards, and other problems [4]. The objective of an inspection is to:

- verify that the software element(s) satisfy its specifications,
- verify that the software element(s) conform to applicable standards,
- identify deviation from standards and specifications, and
- collect software engineering data (such as defect and effort data).

In addition to detecting defects in a software document and thus improving quality, inspecting a software document in a systematic manner teaches the developers to build better software [18].

Inspection techniques that use a non-systematic way of identifying defects are called *ad hoc techniques* [4]. The inspectors must utilize their own experience and knowledge to identify defects. The results of this technique depend solely on the abilities of the inspectors.

In *checklist-based techniques* the inspectors are provided with a list of general defect classes to check against. This kind of inspection technique is most common in industry [6], but the technique has some shortcomings that are described in [11].

A *review* is defined as a manual process that involves multiple readers checking a document for anomalies and omissions [19]. It is generally recommended that representatives of the different stakeholders in a project should participate in the review, but that they should look for different problems and defects.

A *walkthrough* is a peer group review of a software document [21]. It involves several people, each of whom plays a well defined role. A typical walkthrough involves at least one person, most often usually the author, whose job it is to introduce the document to the rest of the group.

In the context of requirements documents, inspections are recommended for defect detection, reviews for consensus and walkthroughs for training [8]. It is also recommended that an inspection should be performed before the two other activities to remove defects, which are noise in the process of achieving consensus on the requirements and in the walkthrough process.

### 2.2 Inspections of Requirements Specifications

The problems with ad hoc and checklist-based inspection techniques have been attempted remedied by introducing a scenario-based technique [14], where a checklist is used as a starting point for a more elaborate technique. The elements of the checklist were replaced by scenarios implementing the elements. The claims for the scenario-based technique is that it teaches the inspectors how to read the requirements documents in order to detect defects, and it offers a strategy for decomposition enabling each of the inspectors to concentrate on distinct parts of the requirements document. The scenario-based technique proved more effective than ad hoc and checklist-based inspections [14]. However, several replications of this evaluation have been conducted with varying results [12,13,15]. The replication reported in [15] found weak support for the original results, while the two other replications did not find the scenario technique superior to the two other techniques.

Different alternative decomposition strategies have been attempted to give the inspectors distinct responsibilities. One strategy is used in perspective-based reading. This technique is based on the classification of defects according to the perspectives represented by the different stakeholders in the project. The perspectives should be tailored to the needs of the various stakeholders, typical perspectives are clients or end-users, developers and testers. The reading technique for the user perspective involves constructing a use case model from the textual requirements.

Another strategy is used in the inspection technique usage-based reading, where a prioritized use case model is taken as input [20]. Its strength is claimed to be that it makes the inspectors focus on the defects that are important for the future users of the system.

## 3 Defects in Use Case Models

To develop and evaluate an inspection technique for use case models, we need knowledge of typical defects in use case models and of their consequences. Table 1 shows our proposal for a taxonomy of defects in use case models. The defects are divided into omissions, incorrect facts, inconsistencies, ambiguities and extraneous information [17]. In addition to the general quality issues presented in [2], we have considered different stakeholders to find a comprehensive list of defects.

Clients and end users want to be sure that they get the expected functionality. In terms of use case models this implies the following:

- The correct actors should be identified and described.
- The correct use cases should be identified and should describe how the use case goals [5] are reached. The actors should be associated with the correct use cases.
- The flow of events in each use case should be realistic in that it leads to the fulfillment of the use case goal. The use case descriptions should be easy to understand for users who are unfamiliar with use case modeling so that the

described functionality can be verified. This implies that the use cases should be described at an appropriate level of detail.
- The functionality should be correctly delimited through the use of pre- and post-conditions and variations.

Project managers need to plan software projects. For example, when estimating software projects, use case models can be used successfully [1]. To support the planning:
- the use case model should cover all the functional requirements, and
- all the interactions between the actor and the system that are relevant to the user, both in the normal flow of events and the variations should be described.

Designers will apply use case models to produce an object-oriented design. Therefore:

- the use of terminology should be consistent throughout the use case descriptions, and
- the use case descriptions should be described at a suitable level of detail. There should be no details of the user interface or internal details that put unnecessary constraints on the design

Testers will apply use case models to test that the functionality is correctly implemented. Therefore:
- the prerequisites (pre-conditions) for the execution of a use case, and the outcome (post-conditions) of each use case should be testable, and
- all terms in the use case descriptions should be testable.

A use case model consists of a diagram that gives an overview of the actors and the use cases, and of textual descriptions of each use case detailing out the requirements, typically using a template [5]. Use cases can, however, be described using many different formats [9]. The actual format may have an impact on the ease of detecting certain defects. For example, it should always be clear what are the pre- and post-conditions, of a use case. If a template format is used, pre- and post-conditions will usually be easily detectable. If the use cases are described with free text, on the other hand, it may be necessary to search the use case description for the information.

Some defects may also be specific to the format. To verify that applicable standards are followed, the inspection technique must be tailored to the actual format used. The proposed taxonomy is based on a format with normal flow of events and variations as well as the use case starting condition (trigger), and pre- and post-conditions. There are both simple and elaborate variants of the template format. We have chosen the simple template since our aim is to present a basic taxonomy that can be further extended to fit an actual project.

**Table 1. Taxonomy of defects in use case models**

| | **Actors** | **Use cases** | **Flow of events** | **Variations** | **Relation between use cases** | **Trigger, pre- and post-conditions** |
|---|---|---|---|---|---|---|
| **Omissions** | Human users or external entities that will interact with the system are not identified | Required functionality is not described in use cases. Actors have goals that do not have corresponding use cases | Input or output for use cases is not described. Events that are necessary for understanding the use cases are missing | Variations that may occur when attempting to achieve the goal of a use case are not specified | Common functionality is not separated out in included use cases | Trigger, pre- or post-conditions have been omitted |
| **Incorrect facts** | Incorrect description of actors or wrong connection between actor and use case | Incorrect description of a use case | Incorrect description of one or several events | Incorrect description of a variation | Not applicable | Incorrect assumptions or results have led to incorrect pre- or post-conditions |
| **Inconsist-encies** | Description of actor is inconsistent with its behavior in use cases | Description is inconsistent with reaching the goal of the use case | Events that are inconsistent with reaching the goal of the use case they are part of | Variations that are inconsistent with the goal of the use case. | Inconsistencies between dia-gram and descriptions, inconsistent terminology, inconsistencies between use cases, or different level of granularity | Pre- or post-conditions are inconsistent with goal or flow of events |
| **Ambiguities** | Too broadly defined actors or ambiguous description of actor | Name of use case does not reflect the goal of the use case | Ambiguous description of events, perhaps because of too little detail | Ambiguous description of what leads to a particular variation | Not applicable | Ambiguous description of trigger, pre- or post-condition |
| **Extraneous information** | Actors that do not derive value from/provide value to the system | Use cases with functionality outside the scope of the system or use cases that duplicate functionality | Superfluous steps or too much detail in steps | Variations that are outside the scope of the system | Not applicable | Superfluous trigger, pre-or post-conditions |
| **Consequences** | Expected functionality is unavailable for some users or interface to other systems are missing | Expected functionality is unavailable | Too many or wrong constraints on the design or the goal is not reached for the actor | Wrong delimitation of functionality | Misunder-standings between different stake-holders, inefficient design and code | Difficult to test the system and bad navigability for users between different use cases |

**1. Actors**

1.1. Are there any actors that are not defined in the use case model, that is, will the system communicate with any other systems, hardware or human users that have not been described?

1.2. Are there any superfluous actors in the use case model, that is, human users or other systems that will not provide input to or receive output from the system?

1.3. Are all the actors clearly described, and do you agree with the descriptions?

1.4. Is it clear which actors are involved in which use cases, and can this be clearly seen from the use case diagram and textual descriptions? Are all the actors connected to the right use cases?

**2. The use cases**

2.1. Is there any missing functionality, that is, do the actors have goals that must be fulfilled, but that have not been described in use cases?

2.2. Are there any superfluous use cases, that is, use cases that are outside the boundary of the system, do not lead to the fulfilment of a goal for an actor or duplicate functionality described in other use cases?

2.3. Do all the use cases lead to the fulfilment of exactly one goal for an actor, and is it clear from the use case name what is the goal?

2.4. Are the descriptions of how the actor interacts with the system in the use cases consistent with the description of the actor?

2.5. Is it clear from the descriptions of the use cases how the goals are reached and do you agree with the descriptions?

**3. The description of each use case**

3.1. Is expected input and output correctly defined in each use case; is the output from the system defined for every input from the actor, both for normal flow of events and variations?

3.2. Does each event in the normal flow of events relate to the goal of its use case?

3.3. Is the flow of events described with concrete terms and measurable concepts and is it described at a suitable level of detail without details that restrict the user interface or the design of the system?

3.4. Are there any variants to the normal flow of events that have not been identified in the use cases, that is, are there any missing variations?

3.5. Are the triggers, starting conditions, for each use case described at the correct level of detail?

3.6. Are the pre- and post-conditions correctly described for all use cases, that is, are they described with the correct level of detail, do the pre- and post conditions match for each of the use cases and are they testable?

4. **Relation between the use cases:**

4.1. Do the use case diagram and the textual descriptions match?

4.2. Has the include-relation been used to factor out common behaviour?

4.3. Does the behaviour of a use case conflict with the behaviour of other use cases?

4.4. Are all the use cases described at the same level of detail?

**Fig. 1. Checklist for inspections of use case model**

## 4 An Inspection Technique for Use Case Models

The checklist approach was chosen as a starting point for developing an inspection technique for use case models, despite the problems mentioned in Section 2, because several such checklists already exist [3,10,16,22]. Checklists were also the starting point for more elaborate inspection techniques for other software documents as described in Section 2.2. In this paper, we have chosen the term inspection instead of

the term review because our focus is on detecting defects rather than on reaching consensus on the requirements. Based on the taxonomy in Section 3 and several recommendations for checklists for use cases models, we developed the checklist in Figure 1.

Our aim was a basic inspection technique which would be generally applicable. The checklists proposed in [3,10,16,22] contain some aspects that we have not included in our checklist because they were considered too specialized for our purpose and applicable only for some projects.

In [3] it is recommended to consider how a use case model fits with the overall business process model. For each use case it should be clear which business event initiates it, and which source it originates from.

The approach described in [10] differs from ours in that it recommends that a review should verify that the use cases meet technical criteria and that the user interfaces are consistent. They recommend that use case granularity should be verified. This is done by asking whether the use case model would be easier to understand if some use cases were split, and whether one path through a use case can be implemented in one iteration in the development project.

Separate reviews for completeness and for potential problems are recommended in [16]. The review for completeness should verify that the use cases fit the architecture and that the user interface matches the use cases. The review for potential problems should be conducted with clients or end users, and developers. Clients and end users should focus on whether they agree on the assumptions behind the functional requirements. Developers should focus on whether they have sufficient information to start construct the system. In addition to our checks, the checklist proposed in [22] recommends prioritization of the use cases for delivery and classification of their importance.

## 5 Evaluation of the Inspection Technique

To empirically evaluate the proposed inspection technique, two studies were conducted: Study 1 and Study 2. The aim of this evaluation was to investigate to what extent the inspection technique would improve defect detection[1].

### 5.1 Study 1

Study 1 was conducted over two semesters (autumn 2000 and autumn 2001) in the context of an undergraduate course in software engineering. The students were taught use case modeling in two lectures, and had exercises in seminars. The course also included a project where the students were organized in teams and developed a small software system.

The students in the course were in their 3rd or 4th year. A large number, approximately 40%, had part-time jobs as software developers or had previously

1 The material used in the evaluation can be found at http://www.ifi.uio.no/forskning/grupper/isu/forskerbasen

worked with software development. About half of them were familiar with UML and use case modeling, mostly from previous courses; only a couple had applied use case modeling professionally.

#### 5.1.1 Design of Study 1

In the project, the students were organized in teams of clients and developers. Two different systems were developed; each team was clients for one system and developers for the other system. In autumn 2000, 139 students divided into 31 teams either developed a hospital roster management system or a system for conducting opinion polls on the internet. In autumn 2001, 118 students divided into 27 teams either developed a hotel room allocation system or a sales management system. The client teams made informal, textual requirements specifications and handed those over to their developers. The developers then constructed use case models. The pairs of teams also had a couple of meetings to clarify the requirements.

During the autumn 2000, the client teams wrote an evaluation report on the use case models they had received. Very few defects were reported even though an analysis by the authors of this paper showed that the use case models did contain many defects.

The following year, autumn 2001, we wanted to investigate whether an inspection technique would improve the teams' ability to detect defects. We also wanted to examine whether the different perspectives represented by respectively the clients and the developers would lead to detection of different defects.

The development teams and the client teams conducted inspections using the checklist in Section 4. Each use case model was therefore inspected twice. The client teams were asked to focus on whether the use case model described the expected functionality. The development teams were asked to focus on whether there was enough information to create a good design, and later test that the delivered system was in accordance with the functional requirements. The teams had approximately two weeks available for this task. The teams registered effort spent on the inspections. There was a large difference in effort between the different teams, ranging from 2 to 30 hours, partly because of differences regarding how many of the team members participated in the inspections. Nevertheless, the registered hours showed that the teams were serious about the inspections.

The inspections resulted in reports that described the defects found. These reports were analyzed, and then the use case models were inspected by two people, one of them the first author of this paper. We decided to accept all the defects found by the teams as actual defects. Since the textual requirements specifications were different for all the teams, we considered the students' knowledge of the requirements to be better than ours. The defects were classified according to the categories described in Section 3.

#### 5.1.2 Results from Study 1

Table 2 shows the total number of defects found by the client teams and the development teams distributed by the categories presented in Section 3. The number of defects found by both the client team and the development team are shown in the row marked *common*. The number 3 in the 'Actors' column means that out of the 92

defects concerning actors in the 27 use case models, only 3 were identified by both the client team and the development team of a particular system. The defects found in the final inspection by the first author and one assistant, and not found by neither the client team nor the development team are shown in the row *not found*.

Almost all the teams found defects and suggested corrections. We consider these results as good indications that the checklists helped the teams to detect defects. This is further supported by the fact that we found very few defects that had been missed by the teams.

The results show that the clients found most defects, on average more than twice as many as the developers, and that there were strikingly few common defects. This indicates a large difference between what is considered a defect in a use case model.

**Table 2. Total number of defects detected in the student project**

| | Actors | Use cases | Flow of events | Variat-ions | Relation between use cases | Trigger, pre/post conditions |
|---|---|---|---|---|---|---|
| **Clients** | 60 | 49 | 59 | 37 | 8 | 48 |
| **Developers** | 26 | 17 | 29 | 24 | 3 | 42 |
| **Common** | 3 | 4 | 2 | 7 | 0 | 9 |
| **Not found** | 6 | 8 | 46 | 3 | 5 | 10 |
| **Total** | 92 | 74 | 134 | 64 | 16 | 100 |

The defects found by the clients frequently appeared to be due to expectations regarding functionality of the system that they had not expressed in the informal requirements specifications nor in the meetings with the development team, but which they missed when they read through and inspected the use case model. Many defects found by the developers were actually elements of the functionality that should have been described more precisely, but these weaknesses were not necessarily defects.

The difference in defects found by the clients and developers indicates that an inspection technique based on different perspectives, similar to perspective-based reading for textual requirements [17], may be useful for use case models. It also shows that after the inspection reviews of the use case models involving different stakeholders in the project can be useful in order to reach consensus on the requirements.

### 5.2 Study 2

Two weeks after the inspections were completed in the student project autumn 2001, a controlled experiment was conducted with 45 of the students as subjects. The students volunteered to participate in the experiment

#### 5.2.1 Design of Study 2

The participants received a textual requirements specification for the hospital roster management system which had been implemented in the student project the previous year. The requirements for the system were based on the requirements for an actual system for a Norwegian hospital. These students were unfamiliar with that system.

They received a use case model for the system with several defects inserted by us. These defects were similar to the defects that we had detected when the system was used in the student project the previous year.

Half of the participants received a checklist similar to the one used in Study 1, shown in Section 4. The checklist in this experiment was slightly adapted to suit a context where the participants were unfamiliar with the actual use case model. Therefore, the checklist explicitly asked the participants to read the textual requirements specification and mark possible actors and their goals, that is, possible use cases. The other half was not given any particular inspection technique; they used ad hoc inspection.

The inspections were performed individually. The students made a list of all the defects, and they commented on the use case model when a defect was detected.

The duration of the experiment was three hours. The students were paid to participate. We did not want time to be a constraint on the experiment, so the subjects where given ample time. They were given an extra task after the inspection to keep them busy for three hours, but it was stressed that they did not have to complete the extra task.

The inspected use case models and the lists of defects were analyzed by the same two persons as in Study 1. The defects were classified according to the categories described in Section 3.

#### 5.2.2 Results from Study 2

Table 3 shows that the inspectors who used the checklist found slightly more defects regarding the actors and the use cases than did those using the ad hoc technique. These defects are the most important, and could have had very serious consequences if not detected early in the development process. The inspectors using the ad hoc technique found more defects in the other categories, but overall the difference in the number of defects detected was negligible. However, Figure 2 shows that the difference in time spent on the inspection is significant in favor of the ad hoc approach. Therefore, using the checklist was more time-consuming without leading to more defects being found.

Table 3 further shows that all the subjects found quite a lot of the defects regarding actors, use cases, triggers and pre- or post-conditions. They did not find many of the defects in the flow of events or defects with superfluous or missing variations. This indicates that such errors are difficult to detect without having developed a more thorough understanding of the requirements.

Table 4 shows that the standard deviation was larger in most categories for those using the ad hoc approach, probably because the subjects using the ad hoc approach used more varied strategies for finding defects.

In addition to detecting defects that were deliberately planted in the use case model, most of the inspectors made some suggestions for how the requirements and the use case model could be improved. They also detected some “false” defects, that is, they were not really defects. There was no noticeable difference between the two inspection approaches.

The results indicate that a checklist or a specific inspection technique may not be particularly useful when the inspectors already have good knowledge about the

defects they are expected to find as had the inspectors in this case; they had recently performed similar inspections. On the contrary, experienced inspectors may be more efficient without a checklist. This supports previous work that did not show any particular differences between ad hoc, checklists or scenario-based techniques [4,12,13]. A checklist may, however, be a good means to assure that a task is performed seriously.

**Table 3. Average number of defects found in the experiment**

| | Actors | Use cases | Flow of events | Variat-ions | Relation between use cases | Trigger, pre/post conditions |
|---|---|---|---|---|---|---|
| **Checklist** | 3,0 | 2,0 | 1,0 | 0,6 | 0,4 | 3,7 |
| **Ad hoc** | 2,8 | 1,8 | 1,7 | 1,0 | 0,6 | 4,6 |
| **Actual defects** | 4 | 4 | 5 | 6 | 4 | 10 |

**Table 4. Standard deviation for number of defects found in the experiment**

| | Actors | Use cases | Flow of events | Variat-ions | Relation between use cases | Trigger, pre/post conditions |
|---|---|---|---|---|---|---|
| **Checklist** | 0,8 | 1,1 | 1,0 | 0,7 | 0,6 | 2,4 |
| **Ad hoc** | 1,0 | 1,2 | 1,0 | 1,0 | 0,5 | 3,0 |
| **Actual defects** | 4 | 4 | 5 | 6 | 4 | 10 |

```
Chi-Square = 16,24   DF = 1   P = 0,000

                                    Individual 95,0% CIs
Type        N<=  N>   Median   Q3-Q1  -------+---------+---------+---------
Checklist    5  18    130,0    24,0                                (--+--------)
Ad hoc      18   4     93,5    35,0  (----+--------------)
                                       -------+---------+---------+---
                                             96       112       128
Overall median = 120,0
```

**Fig. 2. Moods Median test on time spent**

### 5.3 Threats to Validity

The taxonomy of defects in use case models presented in Section 3 requires more work to be more complete. A different taxonomy may lead to different results for the proposed inspection technique. There were some defects in the use case models that were difficult to assign to a specific category. Therefore, the distribution of defects in the different categories might have been slightly different if the defects had been categorized differently.

Both evaluations were conducted with undergraduate students on use case models of rather small scale. We may get different results if evaluations are performed with

inspectors who have more experience with use case modeling and inspections. A follow-up experiment with professional software developers is therefore planned.

The size and format of the use case models may have impacted the results. Larger use case models could have made it infeasible for the inspectors to inspect the whole use case model. The experiment reported in [15] shows that the format of the textual requirements may have a larger impact on the inspectors' ability to detect defects than does the inspection technique. However, the template style used in these evaluations is frequently recommended and is a commonly used format [5].

Study 1 shows that the client teams found most defects. These teams may not be representative of typical clients as they were also developers and thus familiar with use case modeling.

Study 2 shows that the checklist-based inspection technique was not more efficient than the ad hoc technique. In this study, the participants were familiar with the checklist and the classes of defects from the student project. The students performing the ad hoc inspections may therefore have used elements of the checklist even though they did not have the checklist available when performing the inspection.

## 6 Conclusions and Future Work

The quality of a systems' use case model is important for the quality of the resulting software product. In this paper we introduced a tentative taxonomy of defects in use case models and a checklist-based inspection technique to detect such defects. The checklist was evaluated in a student project and subsequently in a controlled experiment, also with students.

We presented anecdotal evidence that inspections may be a useful means to improve the quality of use case models because the teams using the checklist in the student project found many more defects than did the teams not using such a checklist. Clients and developers in our studies found very different defects even though they used the same inspection technique. This indicates that different stakeholders should participate in the inspection.

The controlled experiment showed that experienced inspectors were more efficient without using the checklist. Therefore, more work is needed to establish appropriate inspection techniques. The following activities are planned:

- Studies of use case reviews in actual software development projects to investigate how different stakeholders search for and detect defects in use case models.
- Refinement of the taxonomy and the inspection technique. We plan to investigate how the questions can be tailored to the needs of different stakeholders. We also intend to study how the questions best can be phrased in order to provide appropriate strategies for detecting defects in use cases described with different formats.

## Acknowledgements

We thank Kirsten Ribu for help with the analysis. We also thank all the students who took part in the evaluation of the use case inspection technique.

## References

1. Anda, B., Dreiem, H., Sjøberg, D.I.K., and Jørgensen, M. Estimating Software Development Effort Based on Use Cases - Experiences from Industry. UML'2001, Toronto, Canada, October 1-5, 2001, LNCS 2185 Springer-Verlag, pp. 487-502.
2. Anda, B., Sjøberg, D.I.K. and Jørgensen, M. Quality and Understandability in Use Case Models. ECOOP'2001, June 18-22, 2001, LNCS 2072 Springer-Verlag, pp. 402-428.
3. Armour, F. and Miller, G. Advanced Use Case Modelling. Addison-Wesley, 2000.
4. Cheng, B. and Jeffery, R. Comparing Inspection Strategies for Software Requirement Specifications. Proceedings Australian Software Engineering Conference. IEEE Comput. Soc, Los Alamitos, CA, USA, 1996.
5. Cockburn, A. Writing Effective Use Cases. Addison-Wesley, 2000.
6. El Emam, K. and Laitenberger, O. Evaluating Capture-Recapture Models with Two Inspectors. IEEE Trans. on Softw. Eng., Vol. 27(9), pp. 851-864, September 2001.
7. Fagan, M.E. Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal, Vol. 15(3), pp. 182-211, 1976.
8. Gilb, T. and Graham, D. Software Inspection. Addison-Wesley, 1993.
9. Hurlbut, R.R. A Survey of Approaches for Describing and Formalizing Use Cases. Technical Report: XPT-TR-97-03, Expertech, Ltd., 1997.
10. Kulak, D. and Guiney, E. Use Cases: Requirements in Context. Addison-Wesley, 2000.
11. Laitenberger, O., Atkinson, C., Schlich, M. and El Emam, K. An experimental comparison of reading techniques for defect detection in UML design documents. Journal of Systems and Software, Vol. 53(2), pp. 183-204, August 2000.
12. Lanubile.F. and Visaggio, G. Assessing defect detection methods for software requirements inspections through external replication, ISERN-96-01, January 1996.
13. Miller, J., Wood, M., Roper, M. and Brooks, A. Further Experiences with Scenarios and Checklists. Empirical Software Engineering, Vol. 3(1), pp. 37-64, January 1998.
14. Porter, A.A., Votta, L.G. and Basili, V.R. Comparing Detection Methods for Software Requirements Inspections: a Replicated Experiment. IEEE Trans. on Softw. Eng., Vol. 21(6), pp. 563-575, June 1995.
15. Sandahl, K, Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M. and Ohlsson, N. An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections, Empirical Software Engineering, Vol. 3(4), pp. 327-354, December 1998.
16. Schneider, G. and Winters, J. Applying Use Cases – A Practical Guide. Addison-Wesley, 1998.
17. Shull, F., Rus, I. and Basili, V. How Perspective-Based Reading Can Improve Requirements Inspections. IEEE Computer, Vol. 33(7), pp. 73-79, July 2000.
18. Shull, F., Travassos, G.H., Carver, J. and Basili, V.R. Evolving a Set of Techniques for OO Inspections. CS-TR-4070 and UMIACS-TR-, October 1999.
19. Sommerville, I. Software Engineering, 5th Ed. Addison-Wesley, 1996.
20. Thelin, T., Runeson, P. And Wohlin, C. An Experimental Comparison of Usage-Based and Checklist-Based Reading. WISE 2001.
21. Yourdon, E. Structured Walkthroughs. Prentice-Hall, 1989.
22. www.mcbreen.ab.ca/papers/QAUseCases.htm