Use Case Evaluation (UCE): A Method for Early Usability Evaluation in Software Development

Kasper Hornbæk[1], Rune Thaarup Høegh[2], Michael Bach Pedersen[3] and Jan Stage[2]

[1] University of Copenhagen, Department of Computer Science, Universitetsparken 1, DK-2100 Copenhagen, Denmark
[2] Aalborg University, Department of Computer Science, Fredrik Bajers Vej 7, DK-9220 Aalborg East, Denmark
[3] ETI A/S, Bouet Moellevej 3-5, DK-9400 Nørresundby, Denmark
kash@diku.dk, runethh@cs.aau.dk, mbpedersen@gmail.com, jans@cs.aau.dk

Abstract. It is often argued that usability problems should be identified as early as possible during software development, but many usability evaluation methods do not fit well in early development activities. We propose a method for usability evaluation of use cases, a widely used representation of design ideas produced early in software development processes. The method proceeds by systematic inspection of use cases with reference to a set of guidelines for usable design. To validate the method, four evaluators inspected a set of use cases for a health care application. The usability problems predicted by the evaluators were compared to the result of a conventional think-aloud test. About one fourth of the problems were identified by both think-aloud testing and use case inspection; about half of the predicted problems not found by think-aloud testing were assessed as providing useful input to early development. Qualitative data on the evaluators' experience using the method are also presented. On this background, we argue that use case inspection has a promising potential and discuss its limitations.

Keywords: Usability evaluation, use cases, software development

1 Introduction

Usability evaluation has established itself as an indispensable part of the development of interactive computer applications. A broad variety of methods have been proposed to support evaluators in conducting usability evaluations; Dumas [11] and Cockton et al. [7] present overviews of recent developments. Simultaneously, practice appears to change as more development organizations begins to work focused with usability [2].

Despite these developments, most usability work takes place late in the software development process. Deferring usability work in this way ignores the general observation that faults in software, including usability problems, are much cheaper to solve early in the development cycle [2,26]. Identifying usability problems early, however, is difficult with the current software development practice because usability work is usually separated from core software development activities.

Use cases have been suggested as a valuable means for integrating usability engineering directly into the software development process [10,12]. Use cases are often available early in the development of an interactive system, and are relevant both to software development and user interface design. Though scenarios have many of the same characteristics as use cases, use cases are typically considered more specific and detailed than scenarios (e.g., when described as suggested by Cockburn [5]) and form part of many mainstream development methodologies. Therefore, they form a potentially suitable basis for conducting usability evaluation on an early software development product.

In this paper, we present an evaluation method called Use Case Evaluation (UCE) tailored for usability evaluation based on use cases. Our aim is to facilitate identification of usability problems at the point in the development process where the first key use cases are described. This will also help integrating usability engineering into the development cycle. In order to evaluate the effectiveness of this method, we compare it to a baseline think-aloud test. In Section 2, we give an overview of work with early usability evaluation. Section 3 presents the UCE method. In Section 4, we describe a study designed to validate the method; Section 5 describes the results of the study. Section 6 discusses the results in a broader context and Section 7 provides the conclusion.

2 Related Work

The literature on usability evaluation discusses a variety of design products that may be evaluated early, such as prototypes [39], scenarios [14], storyboards [17], and interface specifications [24]. Among these, use cases and scenarios are two prominent approaches with several desirable characteristics. They are often available early in the development of an interactive system. They also form the cornerstone of several development methodologies, such as the Unified Software Development Process [21] or scenario-based development. In particular for use cases, several authors argue that they provide a strong connection between the fields of software development and user interface design [10,35]. Ferré et al. [12] argued that use cases offer a good starting point for integrating usability techniques into a software engineering process, with the additional benefit that they are understandable from both fields. Below we discuss in more detail the literature on use cases and scenarios.

A use case is a description of a system's behavior as it responds to requests from an actor who wants to achieve a particular goal, or, following Bittner and Spence [3], a description of a sequence of events that leads to a system doing something useful. Use cases were developed by Jacobsen [20] and are currently widely employed to capture the functional requirements of a system. They are typically expressed in ordinary language, avoiding technical jargon and description of the internal workings of a system. However, since use cases were introduced, countless variations on how to describe them have been proposed [5]. Constantine and Lockwood [10] described *essential use cases*, which are free of technology and implementation detail. An essential use case describes a complete, meaningful, and well-defined task of interest to the user. Any design decisions, especially those related to the user interface, are

deferred and abstracted. *Real use cases*, as described by Larman [28], contain concrete design suggestions. Cockburn [5] has also proposed variations on how a use case should be written, including variations in degree of formality of the description, the role of illustrations, and the status of non-functional requirements.

A related stream of research focuses on scenarios. Scenarios originate from strategic management methodology and have since spread to human-computer interaction and software engineering [23]. Scenarios offer an early and systematic approach to describe users' work. Part of Carroll's [4] definition of scenarios state that it is a: "… description sufficiently detailed so that design implications can be inferred and reasoned about". Hertzum [15] has portrayed how scenarios have been used by software developers in real-world projects. He found that scenarios offered a meaningful sequence of events and activities to the developers in the initial phases of the analysis and design process, and that they assisted in preserving a real-world and recognizable feel when trying to understand how users work.

Few studies deal with usability evaluation based on scenarios or use cases. Scenario-based evaluation was investigated by Haynes et al. [14] in relation to CSCW, where it helped identify factors that impacted the system's chance of success. A number of authors propose techniques to assess the quality of use cases. Tao [37] proposed an approach based on defining a behavioural model, expressed as a state machine, which focuses on the flow of interaction between the user and the system as the user is carrying out a task. He suggests usability principles that can be used as guidance for evaluating the behavioural model. Anda and Sjøberg [1] identified typical defects in use cases and present a checklist for finding such defects. In addition, they proposed an inspection technique based on the checklist. The defects are general and deal with unclear or incorrectly expressed use cases, and not with specific usability problems. Jagielska et al. [22] also worked with assessment of use cases. They saw uses cases as expressions in natural language, and based on an analysis of a number of use cases, they suggested guidelines for writing use cases that are easier to understand. Van der Poll et al. [38] employed use case maps to check formal properties of use cases, but only assessed completeness and consistency. Thus the focus is mainly on the use case itself and not on the system it describes.

3 Use Case Evaluation

Usability evaluation with the Use Case Evaluation (UCE) method consists of three overall activities, see Fig. 1: (1) Inspection of use cases, (2) Assessment of use cases, and (3) Documentation of evaluation.

The input for inspection is a collection of *use cases* describing the use of the system under development and a brief description of the use context for the system. We recommended the fully dressed form of use case description proposed by Cockburn [5]. With fully dressed use cases, evaluators receive as much information as possible with a use case. In addition, a list of *guidelines* is required to assist the inspection. The *evaluation product* is an assessment of the usability of the system expressed as a list of usability problems. It may also include an assessment of the

quality of the use cases. The evaluation product is subsequently fed back into the interaction design activity [18].

3.1 Guidelines for Inspection

UCE provides a set of usability guidelines that we suggest as being particularly apt for use case inspection, see Table 1. The guidelines were originally derived from previous research. Our aim has not been to make the guidelines non-overlapping, but merely to provide rich and varied support for identifying usability problems.

UCE is based mainly on heuristics introduced as part of Heuristic Evaluation [30,31]. These heuristics were chosen because they have shown to be applicable across a wide range of contexts, and because they are among the most extensively validated inspection guidelines. Some of these heuristics do, however, concern details of the user interface design that will typically not be specified at the time use case evaluation is likely, for instance the heuristics 'Aesthetic and minimalist design' and 'Help and documentation'. Therefore, they have been excluded from our list.

We have supplemented the heuristics with guidelines from other methods that help focus on other usability concerns, or provide better examples or more detail (see Table 1). Guideline 9 is based on a principle from Cognitive Dimensions [13], somewhat similar to an Ergonomic Criteria [13] on designing for low workload. We also used Cognitive Dimensions [13] to add guideline 10 on premature commitment. Premature commitment occurs when software requires users to do an action or supply information that they are not ready to do or supply. It is related to a principle in Cognitive Walkthrough [40]. Guideline 11 was motivated by the idea that early evaluation on use cases should help establish the utility of the proposed functionality. In particular, we wanted to allow evaluators to focus on issues like task relevance and missing functionality. It has been suggested that evaluators might be less attentive to utility issues than to issues of convenience and to surface-level interface issues [33].

3.2 Inspection of Use Cases

With UCE, the main activity is to inspect the use cases for usability problems. The aim is to identify usability problems that the evaluator is convinced one or more prospective users will experience. A *usability problem* is (cf [29])

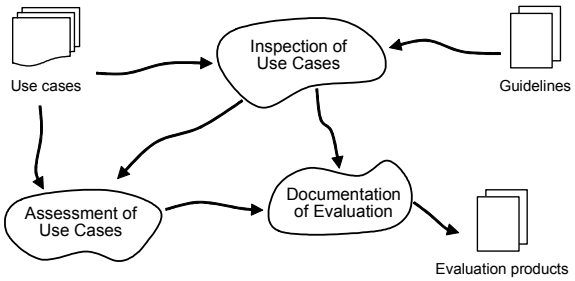

Fig. 1. Overview of Activities and Materials in Use Case Evaluation (UCE).

an aspect of the system that will hinder or delay the user in completing a task, be difficult or impossible for the user understand, or cause the user to be frustrated.

The inspection activity is conducted by one or more evaluators. It involves the following two steps: Brainstorm and Systematic Inspection. In the first step, Brainstorm, the evaluator goes through the use cases one by one without following any systematic procedure. The evaluator notes the usability problems that may be predicted from the use cases.

In the second step, Systematic Inspection, the evaluator employs a structured procedure for inspection of use cases. This procedure follows the one proposed in the early papers on Heuristic Evaluation [30,31]. It is supported with the guidelines presented above (see Table 1). The inspection typically lasts one to two hours. The use cases are inspected one by one. For each use case, the evaluator tries to predict the usability problems a user will experience while carrying out the use case. In doing this, the evaluator employs the guidelines. The aim is to couple the ideals of the guidelines to the use cases, trying to see similarities and cases where a guideline may be breached. As recommended for heuristic evaluation, it is fruitful to go over all use

Table 1. Guidelines for Use Case Evaluation.

No	Guideline	Explanation	Sources
1	Visibility of system status	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time	[31]
2	Match between system and the real world	The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions and make information appear in a natural order.	[31]
3	User control and freedom	Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.	[31]
4	Consistency and standards	Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.	[31]
5	Error prevention	Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Eliminate error-prone conditions or handle them gracefully.	[31]
6	Recognition rather than recall	Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another.	[31]
7	Flexibility and efficiency of use	Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.	[31]
8	Help users recognize, diagnose, and recover from errors	Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.	[31]
9	Avoid hard mental operations and lower workload	Do not force the user into hard mental operations and keep the user's workload at a minimum.	[13,31]
10	Avoid forcing the user to premature commitment	Do not force the user to perform a particular task or decision until it is needed. Will the user know why something must be done?	[13,40]
11	Provide functions that are of utility to the user	Consider whether the functionality described is likely to be useful to users and whether functions/data are missing.	[33]

cases at least twice. The evaluator produces a list of usability problems; if more than one evaluator inspects the interface the evaluators' lists are merged into one joint list.

3.3 Assessment of Use Cases

A secondary activity is to assess the quality of the use cases. In contrast to heuristic evaluation of a fully functional system, there is likely to be several cases where a use case does not give an evaluator sufficient information to decide whether or not a guideline is breached. In those cases, supplementing information may need to be provided or the evaluator may simply express why something cannot be properly analyzed. Sometimes, the use cases will not be specific enough to allow evaluation. In other cases, they will be too specific, e.g., by specifying user interaction details that can be decided later in the development process. Thus the outcome is an assessment of each use case, which emphasizes how useful the use case is for inspection. In practice, this activity is done in parallel with the primary evaluation activity.

3.4 Documentation of Evaluation

In this activity, the results are compiled into a coherent evaluation product that may be fed back into interaction design. The main content of the documentation is the list of usability problems. This list describes problems that the evaluators expect a prospective user will face when using the system. Each of these predicted usability problems should include a clear reason why it is perceived to be a problem. For example, the reason could be a reference to a guideline. As with heuristic evaluation, evaluators should describe only one problem at a time and be as specific as possible. Finally, we suggest that evaluators note ideas for improving the system or designing it differently. A supplementary evaluation product is the assessment of each use case.

4 Empirical Study

The method described above needs to be assessed with respect to its usefulness for use case evaluation. Thus we conducted an empirical study aimed at (a) comparing usability problems identified with the UCE method to a set of problems discovered with think-aloud testing and (b) understanding what problems evaluators experience in using UCE and which guidelines that are particularly useful for inspecting use cases. The rationale for (a) is to investigate if the UCE method find problems that appear in a final version of a system. The rationale for (b) is to obtain input that may be used to improve early us case inspection with the UCE method.

4.1 Evaluators

We had four evaluators use the UCE method to evaluate use cases; two of them are among the authors of this paper. We chose to use four evaluators because it conforms

to the often recommended, though controversial, number of evaluators for inspection [e.g., 32], and because it seemed a realistic number of evaluators to find when bringing UCE to practical use. The evaluators were all experienced with usability work (from two to eight years of experience) and all had conducted several think-aloud tests and usability inspections. Two of the evaluators had a master degree in computer science and were working in industry, one was a PhD student working with usability evaluation, and one was associate professor in human-computer interaction.

4.2 System and Use Cases for Evaluation

The system under evaluation is a health care application (here called the HealthMonitor) that monitors elderly persons' medical conditions in their homes. For instance, it might be needed to monitor their weight, blood glucose level, or blood pressure. The HealthMonitor assists in transmitting the results through a telephone line to a central server, where they can be interpreted by medical staff. If a measurement in some way alerts the medical staff they can decide to recommend the patient to see a doctor, adjust the intake of medicine, or some other remedial action. The HealthMonitor requires connection to the telephone and interfaces with measuring devices relevant to the elderly persons' health conditions. These measuring devices might be a bathroom scale or a device for measuring blood pressure.

A set of four use cases was described for the HealthMonitor. The use cases describe situations where users of the HealthMonitor set up the hardware, use it with different measuring devices, and transfer relevant data to a central server. The use cases were written in the style of Cockburn [5] as fully dressed use cases, meaning that they were semi structured with sections on Goal in Context, Successful End Condition, Primary Actors, and a Main Success Scenario (see Fig. 2 for an example). The four use cases were an average of 472 words long and their main scenarios consisted of 6 to 19 steps. Though the use cases were not taken from the actual development activity (because no use cases had been created originally), they were validated through review by a group of master thesis students who had worked with the HealthMonitor for half a year.

Use case 02: Registration of blood glucose measurement (cabled connection)

CHARACTERISTIC INFORMATION

Goal in Context: The user wants to transfer the result of his newly taken blood glucose measurement from the blood glucose measurement device to the HealthMonitor.
Scope: Enterprise
Level: Summary
Preconditions: A successfully conducted blood glucose measurement. A successfully installed HealthMonitor (Use case 01)
Success End Condition: The blood glucose measurement result is successfully transferred to the HealthMonitor
Failed End Condition: The blood glucose measurement result is not transferred to the HealthMonitor
Primary Actor: A person with a physical health condition that needs daily monitoring.
Trigger: The user wants to register a blood glucose measurement

MAIN SUCCESS SCENARIO

1. The step-by-step instruction for transferring blood glucose measurements from the measuring device to the HealthMonitor is found
2. The two devices is connected using the relevant cable
3. The blood glucose measurement device identifies the connection, and shows the message "PC" in the display
4. The HealthMonitor shows in the display that a transfer is ongoing
5. The HealthMonitor shows in the display the amounts of measurement results that were available in the blood glucose measurement device, and the HealthMonitor also shows how many of those measurement results that have been transferred. The information on the display of the HealthMonitor is further more read aloud to the user by the HealthMonitor.

Fig. 2. Excerpt of Use Case for Connecting a Blood Glucose Device to the HealthMonitor.

4.3 Procedure for Evaluation and Matching of Problems

The evaluations were conducted individually. The evaluators received descriptions of (a) the procedure to be followed, including an instruction on how to describe usability problems, (b) the UCE method, corresponding to the contents of Section 3, (c) the four use cases and an explanation of how to read them, and (d) an explanation of the HealthMonitor's general use context, target users, and basic aims.

To document the evaluation, each usability problem was to be reported by specifying its title and the place or places in the use cases that aided in predicting the problem. The evaluators also assigned a severity rating to each problem, choosing from among three ratings [29]: *cosmetic* (1), meaning that the user is delayed less than one minute or becomes slightly annoyed; *serious* (2), meaning that the problem would delay the user for several minutes or present information or options that to some extent differ from the user's expectations; and *critical* (3), meaning for instance that the user would be unable to continue or become strongly annoyed.

Previous studies [e.g., 8,19] have suggested that studying the process of evaluation may give rich information about benefits and drawbacks of inspection methods and their use. Consequently we adapted the extended reporting format proposed by Cockton et al. [9] by requiring evaluators to answer questions about the manner in which a problem was discovered, the guidelines used to predict a problem (and a comment on why a guideline was used), and if something initially considered a usability problem was excluded from the final problem report. Furthermore, we asked the evaluators to comment on difficulties in using a use case to predict a problem.

After the evaluators had completed their evaluations they met to match the problems, that is, to agree on which problems that were similar and which were unique. A total of nine hours were used on matching. After matching, each evaluator checked that the matching was correct in their view and that they agreed on how problems were treated. Below we refer to problems found by individual evaluators as *problem tokens* and matched problems as *problem types*.

4.4 Comparison to Usability Problems Found with Think-Aloud Testing

Problems predicted with UCE were compared to a set of usability problems found by think-aloud testing with the aim of discussing which problems would be useful to predict. The think-aloud test was conducted by one of the authors of this paper (who did not inspect the use cases) and five students working on a master thesis in human-computer interaction. The system was tested in five user sessions, each lasting about one hour. The sessions were based on predefined tasks that covered the same areas as the four use cases mentioned above. The sessions were recorded on video. Two of the students and the author did an Instant Data Analysis [27] immediately following each test session. This resulted in one list of usability problems. The other three students conducted a conventional video analysis with transcripts and log files to generate another list of usability problems. These two methods were used for analysis to identify as rich a problem set as possible. The resulting two lists of usability problems were merged by the evaluators through negotiation. The resulting common list

included 54 usability-problem types. Below we refer to these problems as think-aloud, or TA, problems.

Similarly to John and Marks [25], we further analyzed the overlap among problems types found by the two evaluation methods. The intuition is as follows (see Table 2). For a problem found with TA but not found with UCE two possibilities exist. One is that a problem could simply not be predicted from the use case, for example because the problem concerned user interface issues decided on during implementation, difficulties with manuals and support, or performance issues (we call these problems *impossible or hard to predict*). Another possibility is that the problem was predictable but missed by UCE (*predictable but missed*). A similar analysis can be made for usability problems predicted by UCE but not found with TA. For such problems we distinguish three possibilities. First, a predicted problem may represent an actual problem or a sensible concern about usability related issues (what we call a *relevant problem*). This happens for example because a possibility for error has not been removed or because of a lack of feedback. Second, a predicted problem could have become a problem, but the actual design of the HealthMonitor has avoided the problem (a *problem avoided*). This happens for instance when feedback is given in the actual interface or when an unclear action is better explained. Thus, raising the problem as a concern early in the design process would have been valuable. Third, a predicted problem may not be an actual problem or legitimate design concern (i.e., essentially of no value as feedback or *not a problem*). An example is problems that merely speculated about a potentially difficult, but highly improbable, use situation. Each problem was rated individually by the authors on the above dimensions and disagreements were settled through discussion.

5 Results

We characterize the results of the evaluation by first discussing the problems predicted by UCE, then discussing the overlap to think-aloud testing, and finally summarizing the evaluators' experiences with using the method.

5.1 Problems Predicted

Using UCE each evaluator predicted an average of 23 problem tokens (*SD* = 5.6, ranging from 18 to 31 predictions). The average severity of the problem tokens predicted were 1.75 (*SD* = 0.79). The matching of problems allowed a set of problem types predicted by UCE to be formed, comprising 61 types. This problem set shows that evaluators have relatively limited overlap with each other: the number of unique problems (i.e., problems predicted by just one evaluator) is on average 5.25 (*SD* = 3.69). Two problem types (covering 3% of all problem tokens) were identified by all evaluators, 5 types (8%) by three evaluators, and 18 (30%) by two evaluators. Another way of illustrating this is to calculate the any-two agreement, suggested by Hertzum and Jacobsen [16] as a indicator of the evaluator effect. By that measure, the average agreement among evaluators is 18.9%. While this number is within the range

Table 2. Problems predicted with the use case inspection (UCE) in relation to those found in think-aloud tests (TA). Percentages are relative to the total number of problem types (93).

Predicted with UCE	*Found with TA*	*Number of problem types*	*Problem category*	*Number of problem types in category*
YES	YES	22 (24%)	-	-
NO	YES	32 (34%)		
			Impossible or hard to predict	20 (21%)
			Predictable but missed	12 (13%)
YES	NO	39 (42%)		
			Relevant problem	14 (15%)
			Problems avoided in UI	9 (10%)
			Not a problem	16 (17%)

of evaluator agreement found by Hertzum and Jacobsen, it suggests substantial variation among evaluators.

5.2 Overlap Between UCE and TA

The matching of problem types found by UCE and think-aloud testing allows us to calculate the overlap between methods. As shown in Table 2, we find an overlap of 24%. Thirty-nine usability problems were predicted with UCE but not found by TA; 32 usability problems were found by TA but not predicted by UCE.

Table 2 also shows that about two-thirds of the problems found by TA but missed by USE could not have been predicted. These problems concern unpredictable actions by the user and issues that arise from design solutions at a more detailed level than the use cases. A special instance of non-predictable problems concerns the direct transfer of technical terms from the use cases ("detecting a phone line") to the interface.

The problem types predicted with UCE but not found with think-aloud testing were more difficult to reason about. The bottom part of Table 2 shows that similar proportions of the problems predicted by UCE but not found with TA were seen relevant (14 problems) and not-a-problem (16 problems). Among those problems found relevant, many appear not to have been found by TA because the test setting prohibited them from occurring. Conversely, the problems that seemed irrelevant often assumed quite intricate and improbable contexts of use. This type of predictions from inspection techniques has previously been noted [7]. These analyses show that overall 45 of the 61 UCE problems (74%) would be useful to know (22 problem types found by TA, 14 problem types that seen as relevant problems, 9 problem types avoided in the user interface). If we consider just problems predicted by UCE but not found with TA, 59% appear useful.

In order to learn more about the usability problems found with UCE, all problem types were analyzed with a simple grounded theory approach [36]. The result of the analysis was five overall areas where usability problems were found. The five areas are presented in the Table 3, together with the number of problems each method identified. Table 3 indicates that there is a large overlap in the types of usability

problems the two methods identifies. Looking at the specific problems, however, shows that the problems found by TA concern issues directly related to the evaluation tasks and the test setup, whereas UCE also predicts problems that are unlinked to the evaluation task and setup, such as problems related to breakdowns that were not a part of the test setup.

5.3 Evaluators' Experiences and Use of Guidelines

Evaluators used between 2.5 and 5 hours on preparing, conducting and reporting the evaluation (M = 3.65 hours, SD = 1.35). Time was mainly used for performing the actual evaluation (about two hours) and reporting the problems (about one hour).

Evaluators' comments on the process of evaluating give a couple of insights. Three evaluators commented on the role of imagination during the inspection. They suggested that evaluating use cases is demanding because they require the evaluator to be creative in filling in the many details that are not described in the use cases. Further, with HealthMonitor, imagining the physical appearance of the device was difficult. One evaluator noted that "when running through the use cases the understanding of how the devices look physically becomes an exercise in subjective imagination. This imagination may be way off the intended product".

Two evaluators commented that many predicted problems might or might not turn out to be problems when the system was implemented. One of them mentioned the feeling that "one often is describing 'maybe problems'". Another evaluator mentioned that predicted problems might turn out not to be problems after all; in the description of several of the predicted problems that evaluator listed good reasons why something might *not* be a problem after all. Also, one evaluator noted that predictions may only concern the writing of a use case, not how that use case ends up being implemented.

Two evaluators noted that it was somewhat difficult to use the guidelines during inspection. One of the evaluators noted that "I don't think I would have found fewer [problems] without them [the guidelines]".

Finally, one evaluator pointed out that it was difficult to predict problems concerning the relation among use cases, because the evaluation procedure only required use cases to be considered individually, not in concert. Problems concerning consistency between use cases, for instance, would rarely be reported. From evaluators' reports it is possible to describe which guidelines are used most frequently. The most frequently used guidelines are 1, 5 and 2; these guidelines are mentioned in 30%, 18%, and 15% of the problems that describe the guideline(s) used

Table 3. Usability problems categories distributed by areas (N = 93). Note that a problem may be related to several areas, so column sums are higher than the number of problems.

Types	*Total*	*Found with TA*	*Found with UCE*
Dialogue	38	25	30
External factors	4	1	3
Graphical User Interface	27	15	20
Installation of equipment	38	29	19
Procedure / task flow	38	18	26

for discovery. These numbers corroborate the findings about problem types. Though all guidelines were used, guidelines 9, 6, and 3 were each used in 2% or fewer cases.

6 Discussion

Our study shows that the UCE method for inspection of use cases made it possible to predict a large portion of the usability problems identified in a conventional think-aloud test of the system. With UCE, the evaluators were able to find 22 usability problems out of the 54 problems found with the conventional approach, and they were able to describe a broad variety of usability problems in detail. Further, many of the predicted problems not seen with the conventional evaluation approach are assessed as being useful. These results suggest that it is feasible and of use to conduct usability inspections on use cases.

The fundamental idea of UCE, to evaluate the usability based on a design product that is available early, could have additional important benefits. First, the inspection of use cases may introduce thinking about usability early and naturally in the software development process. Second, the inspection of use cases may uncover and emphasize non-functional requirements that can be added to the use cases. Third, the inspection may improve the overall quality of the use cases. These benefits, however, remain to be empirically documented.

The idea of UCE is inherited from the group of usability evaluation methods commonly referred to as inspection or walkthrough methods. An essential limitation of for these methods is the identification of false positives. This refers to the problems that are found in an inspection but not in a think-aloud test. The opposite difficulty is the usability problems that are found with a conventional usability evaluation but not with UCE. Some of these problems can never be found. When this is taken into account, there are only 12 problem types found with the conventional method that were *not* found with UCE.

The data on the use of UCE uncovered potential improvements. They include the style of writing use cases (Cockburn, 2000), the set of guidelines we have developed for the evaluators and a step with inspection across the whole collection of use cases with the purpose of discovering inconsistencies between them.

7 Conclusion

We have presented a method called Use Case Evaluation (UCE) tailored for early usability evaluation. It is generally agreed that usability problems should be identified as early as possible in software development. This is, however, difficult to achieve with conventional usability evaluation methods. The UCE method overcomes this problem by predicting usability problems from inspection of use cases. The advantage of this approach is that use cases are often available early in the development process.

To validate the UCE method, we compare it to a conventional think-aloud test. About one fourth of all problems were found by both think-aloud testing and inspection of use cases. In addition, three-quarters of the total number of predicted

problems was assessed as useful input to early development. We have also collected qualitative data on the evaluators' experience using the method, which indicate that use case inspection requires a lot of imagination.

This paper is based on an empirical study that is limited in a couple of respects. First of all, we did not assess impact of UCE evaluation in a real-life context, with use cases crafted by software developers as part of their activities. Second, the study was not a strict experiment, in that participants were not randomly assigned to either think-aloud testing or to the UCE method. Third and finally, the method needs validation with non-expert participants. Moreover, the study was partly conducted by the authors of this paper, who have also developed the method. Therefore, it is necessary with a follow-up study conducted by other researchers. The inspections were conducted by usability experts. It would also be interesting to explore to what extent less experienced evaluators could carry out the inspection. Despite these limitations, our paper suggests that inspection of use cases may help introduce effective usability evaluation early in software development processes.

Acknowledgments. The research behind this paper was partly financed by the Danish Research Councils (grant number 2106-04-0022). We are grateful to Janne Jul Jensen and Christian Monrad Nielsen for serving as evaluators.

References

1. Anda B & Sjøberg DI (2002). Towards an inspection technique for use case models. Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering (pp. 127-134).
2. Anderson J, Fleek F, Garrity K, & Drake F (2001). Integrating Usability Techniques into Software Development. IEEE Software, 18, 1, 46-53.
3. Bittner K & Spence I (2002). Use case modeling. Addison-Wesley.
4. Carroll J (1995). Scenario-based design - envisioning work and technology in system development. John Wiley &. Sons
5. Cockburn A (2000). Writing effective use cases. Addison Wesley.
6. Cockburn A (2002). Use cases, ten years later. STQE Magazine, Mar/Apr, Mar/Apr.
7. Cockton G, Lavery D, & Woolrych A (2003). Inspection-based evaluations. In Jacko JA & Sears A (Eds.), The human-computer interaction handbook (pp. 1118-1138).
8. Cockton G, Woolrych A, Hall L, & Hindmarch M (2003). Changing Analysts' Tunes. Proceedings of People and Computers XVII: Designing for Society (pp. 145-162).
9. Cockton G, Woolrych A, & Hindemarch M (2004). Reconditioned merchandise: extended structured report formats in usability inspection. Proceedings of CHI2004 (pp. 1433-1436).
10. Constantine L & Lockwood L (1999). Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. Addison Wesley.
11. Dumas JS (2003). User-based Evaluations. In Jacko JA & Sears A (Eds.), The human-computer interaction handbook (pp. 1093-1117).
12. Ferré X, Jurisot N, Windl H, & Constantine L (2001). Usability Basics for Software Developers. IEEE Software, 18, 1, 22-29.
13. Green TRG (1989). Cognitive Dimensions of Notations. In Proceedings of People and Computers V (pp. 443-460).
14. Haynes SR, Purao S, & Skattebo AL (2004). Situating evaluation in scenarios of use. In Proceedings of CSCW 2004 (pp. 92-101).

15. Hertzum M (2003). Making use of scenarios: a field study of conceptual design. International Journal of Human-Computer Studies, 58, 2, 215-239.
16. Hertzum M & Jacobsen NE (2001). The evaluator effect: A chilling fact about usability evaluation methods. International Journal of Human-Computer Interaction, 13, 421-443.
17. Holtblatt K, Wendell JB, & Wood S (2005). Rapid contextual design. Morgan Kaufman.
18. Hornbæk K & Stage J (2006). The Interplay Between Usability Evaluation and User Interaction Design. International Journal of Human Computer Interaction, 21, 2, 117-123.
19. Hornbæk K & Frøkjær E (2004). Two psychology-based usability inspection techniques studied in a diary experiment. Proceedings of Nordichi 2004 (pp. 1-8).
20. Jacobson I (1987). Object oriented development in an industrial environment. In Proceedings of OOPSLA '87 (pp. 183-191). New York, NY: ACM.
21. Jacobson I, Booch G, & Rumbaugh J (1999). The unified software development process. Boston, MA: Addison-Weley.
22. Jagielska D, Wenick P, Wood M, & Bennett S (2006). How Natural is Natural Language? How Well do Computer Science Students Write Use Cases? Proceedings of OOPSLA'06 (pp. 914-923).
23. Jarke M, Bui XT, & Carroll J (1998). Scenario management: an interdisciplinary approach. Requirements Engineering, 3, 3&4, 155-173.
24. John BE & Mashyna MM (1997). Evaluating a Multimedia Authoring Tool. Journal of the American Society of Information Science, 48, 9, 1004-1022.
25. John BE & Marks SJ (1997). Tracking the effectiveness of usability evaluation methods. Behaviour & Information Technology, 16, 4/5, 188-202.
26. Juristo N, Windl H, & Constantine L (2001). Introducing usability. IEEE Software, 20-21.
27. Kjeldskov J, Skov M, & Stage J (2004). Instant Data Analysis. Proceedings of the third Nordic conference on Human-computer interaction (pp. 233-240).
28. Larman C (1998). Applying uml and patterns: an introduction to object-oriented analysis and design. Upper Saddle River, NJ: Prentice Hall.
29. Molich R (2005). brugervenligt webdesign. Nyt Teknisk Forlag.
30. Molich R & Nielsen J. (1990). improving a human-computer dialogue. Communications of the ACM, 33(3), 338-348.
31. Nielsen J (1994). Heuristic Evaluation. In Nielsen J & Mack R (Eds.), usability inspection methods (pp. 25-62). John Wiley & Sons.
32. Nielsen J & Landauer T (1993). A mathematical model of the finding of usability problems. Proceedings of CHI 1993 (pp. 206-213).
33. Nørgaard M & Hornbæk K (2006). What Do Usability Evaluators Do in Practice? An Explorative Study of Think-Aloud Testing. In Proceedings of DIS2006.
34. Scapin DL & Bastien JMC (1997). Ergonomic Criteria for Evaluating the Ergonomic Quality of Interactive Systems. Behaviour & Information Technology, 16, 4/5, 220-231.
35. Seffah A, Djouab R, & Antunes H (2001). Comparing and reconciling usability-centered and use case-driven requirements engineering processes. Proceedings of the 2nd Australasian conference on User interface (pp. 132-139).
36. Straus A & Corbett J (1998). Basics of qualitative research. Sage.
37. Tao Y (2005). Developing Usable GUI Applications with Early Usability Evaluation. In Proceedings of Proceedings: Software Engineering - 2005.
38. van der Poll JA, Kotzé P, Seffah A, Radhakrishnan T, & Alsumait A (2003). Combining UCMs and Formal Methods for Representing and Checking the Validity of Scenarios as User Requirements. Proceedings of SAICSIT 2003 (pp. 59-68).
39. Walker M, Takayama L, & Landay JA (2002). High-fidelity or Low Fidelity, Paper or Computer? Proceedings of HFES 2002 (pp. 661-665).
40. Wharton C, Rieman J, Lewis C, & Polson P (1994). The cognitive walkthrough method: a practitioner's guide. In Nielsen J & Mack RL (Eds.), Usability inspection methods (pp. 105-140). John Wiley & Sons.