Use Cases versus Task Descriptions

Soren Lauesen, Mohammad A. Kuhail

IT University of Copenhagen, Denmark
slauesen@itu.dk, moak@itu.dk

Abstract. [Context and motivation] Use cases are widely used as a substantial part of requirements, also when little programming is expected (COTS-based systems). **[Question/problem]** Are use cases effective as requirements? To answer this question, we invited professionals and researchers to specify requirements for the same project: *Acquire a new system to support a hotline*. **[Principal ideas/results]** Among the 15 replies, eight used traditional use cases that specified a dialog between users and system. Seven used a related technique, task description, which specified the customer's needs without specifying a dialog. **[Contribution]** It turned out that the traditional use cases covered the customer's needs poorly in areas where improvement was important but difficult. Use cases also restricted the solution space severely. Tasks didn't have these problems and allowed an easy comparison of solutions.

Keywords: use case; task description; requirements; verification; COTS.

1 Background

Traditional requirements consist of a list of system-shall-do statements, but don't describe the context of use. IEEE-830, for instance, uses this approach [6]. The lack of context makes it hard for users to validate such requirements, and developers often misunderstand the needs (Kulak & Guiney [9]). Jacobson introduced use cases in the late 1980s [7]. They describe the dialog (interaction) between a system and a user as a sequence of steps. Use cases seemed to provide what traditional requirements lacked, and they became widely used as a substantial part of requirements. Soon other authors improved on the basic idea and wrote textbooks for practitioners, e.g. Cockburn [3], Kulak & Guiney [9], Armour & Miller [2], Constantine & Lockwood [4].

Use cases have developed into many directions. Some authors stress that use cases should be easy to read for stakeholders and that they should not be decomposed into tiny use cases [3], [8]. The CREWS project claimed that use cases should be detailed and program-like with *If* and *While*, and have rules, exceptions and preconditions (reported in [1] and questioned in [5]). In contrast, Lilly [15] and Cockburn [3] advise against program-like elements. Other authors claim that use cases must have dialog details to help developers [17], [19].

In this paper we will only discuss how use cases handle the user-system interaction. This is the most common use in literature as well as in practice.

Virtually nobody discusses whether use cases in practice are suited as verifiable requirements. Use cases seem to be intended for system parts to be built from scratch, but this situation is rare today. What if the use case dialog differs from the dialog in one of the potential systems? Should we discard the system for this reason?

As a consultant, Lauesen had observed how use cases were used as a main part of requirements in large software acquisitions, even though the customer expected a COTS-based system with little add-on functionality. Usually the use cases were not used later in the project. However, in one large project the customer insisted on them being followed closely. As a result, the system became so cumbersome to use that the project was terminated [18].

Task Description is a related technique that claims to cover also the case where the system is not built from scratch. One difference is that tasks don't describe a dialog between user and system, but what user and system have to do together. The supplier defines the solution and the dialog - not the customer. The task technique was developed in 1998 (Lauesen [10], [11]) and has matured since [12], [13].

To get a solid comparison of the two approaches, we looked at a specific real-life project, invited professionals and researchers to specify the requirements with their preferred technique, and compared the replies.

2 The Hotline Case Study

The case study is an existing hotline (help desk). Hotline staff were not happy with their existing support system and wanted to improve the one they had or acquire a new one, probably COTS-based.

Lauesen interviewed the stakeholders, observed the existing support system in use, and wrote the findings in a three-page analysis report. Here is a brief summary: The hotline receives help requests from IT users. A request is first handled by a 1st level supporter, who in 80% of the cases can remedy the problem and close the case. He passes the remaining requests on to 2nd line supporters. A supporter has many choices, e.g. remedy the problem himself, ask for more information, add a note to the request, transfer it to a specialist, order components from another company, park the request, and combinations of these. The hotline is rather informal and supporters frequently change roles or attend to other duties.

We invited professionals to write requirement specifications based on the analysis report. The invitation emphasized that we looked for many kinds of "use cases" and didn't care about non-functional requirements. It started this way:

We - the IT professionals - often write some kind of use cases. Our "use cases" may be quite different, e.g. UML-style, tasks, scenarios, or user stories. Which kind is best?

Participants could ask questions for clarification, but few did. The full analysis report is available in [14]. It started this way:

A company with around 1000 IT users has its own hotline (help desk). They are unhappy with their present open-source system for hotline support, and want to acquire a better one. They don't know whether to modify the system they have or buy a new one.

An analyst has interviewed the stakeholders and observed what actually goes on. You find his report below. Based on this, your task is to specify some of the requirements to the new system: use cases (or the like) and if necessary the data requirements.

We announced the case study in June-July 2009 to members of the Requirements Engineering Online Discussion Forum <re-online@it.uts.edu.au>, to members of the Danish Requirements Experience Group, and to personal contacts. The British Computer Society announced it in their July 2009 Requirements Newsletter. We got several comments saying: *this is a great idea, but I don't have the time to participate*. Lauesen wrote special requests to Alistair Cockburn and IBM's Rational group in Denmark, but got no reply.

We received 15 replies. Eight replies were based on use-cases and seven on tasks. Some replies contained separate data requirements, e.g. E/R models, and some contained use cases on a higher level, e.g. business flows. When identifying verifiable requirements, we looked at these parts too. The full replies are available in [14]. Here is a profile of the experts behind the replies:

Replies based on use cases. Decreasing requirement completeness:

Expert A.	A research group in Heidelberg, Germany. The team has industry experience and has taught their own version of use cases for 7 years.
Expert B.	Consultant in California, US. Has 15 years professional requirements experience. Rational-certified for 11 years.
Expert C.	A research group at Fraunhofer, Germany.
Expert D.	Researcher at the IT-University of Copenhagen. Learned use cases as part of his education in UK, but has no professional experience with them.
Expert E.	Consultant in Sweden with ten years experience. Specialist in requirements management. Have classes in that discipline.
Expert F.	A software house in Delhi. Write use cases regularly for their clients. Were invited to write a reply on a contract basis, and were paid for 30 hours.
Expert G.	Consultant in Sweden with four years experience in requirements.
Expert H.	Consultant in Denmark with many years experience. Teaches requirement courses for the Danish IT association.

Replies based on task descriptions. Decreasing requirement completeness:

Expert I.	Consultant in Sweden with many years experience. Uses tasks as well as use cases, depending on the project. Selected tasks for this project because they were most suitable and faster to write.
Expert J.	Help desk manager. Has 15 years experience with programming, etc.
Expert K.	A recently graduated student who has used tasks for 1.5 years. Little program experience.
Expert L.	Researcher at the IT-University of Copenhagen. Has 25 years industry experience and has later used tasks for 10 years as a consultant and teacher.
Expert M.	GUI designer with one year of professional requirements experience.
Expert N.	Leading software developer with 12 years experience. Learned about tasks at a course and wrote the reply as part of a four-hour written exam.
Expert O.	A research group in Bonn. Germany. The team has several years of industry experience, but little experience with the task principle.

3 Evaluation Method and Validity

We evaluated the replies according to many factors, but here we deal only with these:

A. Completeness: Are all customer needs reflected in the requirements?

B. Correctness: Does each requirement reflect a customer need? Some requirements are incorrect because they are too *restrictive* so that good solutions might be rejected. Other requirements are incorrect because they are *wrong*; they specify something the customer explicitly does not want.

C. Understandability: Can stakeholders understand and use the requirements?

Validity threats. For space reasons we don't discuss all the validity threats here, but only the most important ones:

- Lauesen has invented one of the techniques to be evaluated. His evaluation of the replies might be biased. We have reduced this threat by getting consensus from several experts, as explained in the procedure below.
- The experts didn't have the same opportunities for talking to the client, as they would have in real life. True, but it seemed to have little effect. During the consensus procedure, there was no disagreement about which requirements were justified by the analysis report and which were not.
- Lauesen studied the domain and wrote the analysis report. This gave him an advantage. True, but it had no influence on the other six task-based replies. Further, when ranking the task-based replies on completeness (Figure 5), Lauesen (Expert L) was only number 4. (His excuse for the low ranking is that he spent only one hour writing the solution, plus 5 hours pretty-typing it without improving it in other ways.)

Procedure

1. The two authors have different backgrounds and independently produced two very different replies, Kuhail's based on use cases (Expert D) and Lauesen's on tasks (Expert L). We evaluated all replies independently. Each of us spent around 1 to 3 hours on each reply.
2. We compared and discussed until we had consensus. As an example, one of us might have found a missing requirement in reply X, but the other could point to where the requirement had been stated in the reply. We had around 5 points to discuss for each reply.
3. For each reply, we sent our joint evaluation to the experts asking for comments and for permission to publish their reply and our comments. We also asked for comments to our own solutions. Some authors pointed out a few mistakes in our evaluation, for instance that we had mentioned missing requirements in their reply that were not justified by the analysis report; or that our own solutions missed more requirements than we had noticed ourselves. We easily agreed on these points. Other authors said that our evaluation basically was correct, and that they were surprised to see the task approach, and considered using it in the future.
4. Finally we asked two supporters (stakeholders) to evaluate the four representative replies presented below. It was of course a blind evaluation. They had no idea about the authors.

We asked the supporters these questions in writing:

a. *How easy is it to understand the requirements?*
b. Which requirements are covered [met] *by the system you have today?*
c. Which requirements specify something you miss today?[want-to-have]
d. Do you miss something in addition to what is specified in the requirements?
e. Are some requirements wrong?
f. Could you use the requirements for evaluating the COTS system you intend to purchase?

The supporter got an introduction to the case and the style used in the first reply. He/she was asked to read it alone and answer the questions above. He/she spent between 30 and 50 minutes on each reply. Next we met, looked at the reply and asked questions for clarification. We handled the other replies in the same way.

Karin, senior supporter: The first supporter was Karin Tjoa Nielsen. Karin had been the main source of information when Lauesen wrote the analysis report. She is not a programmer but has a very good understanding of users' problems and the hotline procedures. She read the replies in this sequence: Expert H (tiny use cases), Expert L (Lauesen tasks), Expert A (program-like use cases), Expert K (tasks).

Morten, supporter with programming background: The second supporter was Morten Sværke Andersen. Morten is a web-programmer, but had worked in the hotline until two years ago. He had never seen use cases, but had worked with user stories. He read the replies in this sequence: Expert L (Lauesen tasks), Expert H (tiny use cases), Expert A (program-like use cases), Expert K (tasks).

4 Sample Replies and Stakeholder Assessment

We will illustrate the replies with the four examples summarized in Fig. 1. We chose them because they are short, well-written examples of tiny use cases, program-like use cases, and task descriptions.

Figure 1. Overview of four replies.

Expert H (tiny use cases)	**Expert A** (program-like use cases)	**Expert L (Lauesen)** (tasks - no dialog)	**Expert K** (tasks - no dialog)
UC1. Record new request	UC1. Trigger and control hotline problem solution	T1: Report a problem	T1: IT users (report problem + follow up)
UC2. Follow up on request	UC2. Accept request	T2: Follow up on a problem	
UC3. Add request data	UC3. Clarify request	T3: Handle a request in first line	T2.1: First line
UC4. Transfer request	UC4. Handle request	T4: Handle a request in second line	T2.2: Second line
UC5. Update request	UC5. Set support level	T5: Change role	T2.3: Both lines (change state)
UC6. Retrieve statistics	UC6. Get statistics	T6: Study performance	
UC7. Generate reminder (system-to-system use case)	UC7. Warn about orphaned requests (system-to-system use case)	T7: Handle message from an external supplier	
		T8. Update basic data	
Length: 5500 chars. **Time**: 2.5 hours.	**Length**: 9900 chars. **Time**: Unknown.	**Length**: 4600 chars. **Time**: 6 hours.	**Length**: 2900 chars. **Time**: 3 hours.

Tiny use cases: Expert H's reply consists of seven use cases. Fig. 2 shows one of them in detail (*Transfer request*). It describes the dialog when a supporter wants to transfer a help request to another supporter. The steps alternate between *The system does* and *The user does*. Deviations from this sequence are recorded as variants below the main flow, e.g. the variant that the user wants to filter the requests to see only his own.

Expert H's use cases have a simple flow with few deviations from the main flow. They are examples of *tiny use cases*, where each use case describes a simple action carried out by the user.

The reply is easy to read. However, it gives an inconvenient dialog if the system is implemented as described. As an example, several use cases start with the same steps (UC 1, 2 and 3). If implemented this way, the supporter will have to select and open the request twice in order to add a note to the request and then transfer it (a common combination in a hotline.)

Supporter assessment: The supporters were confused about the supplementary fields with goal, actor, etc. and ignored them as unimportant. Apart from this, they found the reply easy to read, but concluded that it contained only few and trivial requirements. As an example, Morten considered the entire use case in Fig. 2 one trivial requirement. *It is more of a build-specification*, he said. They noticed several missing or wrong requirements.

Figure 2. Expert H: Dialog steps in one column. Tiny use cases.

USE CASE #04	Transfer request
Goal	Transfer a request to a specific hotline employee
Level	User goal (sea level)
Precondition	User is logged in and has the right to transfer requests
Postcondition	N/A
Primary, Secondary actors	Hotline
Trigger	Primary actor

NORMAL FLOW	
Step	**Action**
1	The system shows a list of all open requests
2	The user selects a support request
3	The system shows request data for the selected request
4	The system shows a list of hotline employees
5	The user chooses a hotline employee from the list
6	The system changes Owner to the selected employee and changes state to "2nd level"
7	The system updates the request

VARIANTS	
Step	Action
2a 1 2	**User wants to filter his own requests** The system shows a list of all open requests with the user as Owner The use case goes to step 2

Program-like use cases: Expert A's reply also consists of seven use cases. Fig. 3 shows one of them in detail (*Handle request*). It describes the dialog when a supporter takes on a help request and either corrects the problem or transfers the request to someone else. The steps are shown as two columns, one for the user actions (A1, A2,

Figure 3. Expert A: Dialog steps in two columns. Program-like use cases.

Name	Handle request	
Actor	Supporter (first/second line)	
Supporting Actors	IT user	
Goal	Solve a problem.	
Precondition	[Workspace: request]	
Description	**Actor**	**System**
	A1) VAR1) the actor takes on an open request from his/her line. [Exception: No open requests] VAR2) the actor receives a request forwarded to him/her.	S1) If VAR1) The system records the actor as owner of the request. [System function: take on request]
	A2) [optional *] The actor adds information [Include UC Clarify request]	
	A3) [optional X] The actor forwards the request. VAR1) forward to second line VAR2) forward to specific expert (no matter what line) [include UC Handle request]	S3) The system forwards the request [System function: forward request] If VAR1) The system changes the owner to "not set" and the status to "second line" If VAR2: The system sets the expert as the owner and notifies him/her about the forwarded request. If he/she is logged in at the moment, the system sends an alert in a way designed to attract his/her attention (e.g. a pop-up window). Else it sends the alert as soon as the expert logs in.
	A4) [optional *] The actor looks up information of the request.	S4) The system provides information about the request. [System function: show request details]
	A5) [optional X] The actor solves the problem and closes the request.	S5) The system closes the request. The system sends the user a notification. [System function: close request]
Exceptions	[There are no open requests]: The system contains no open requests.	
Rules	None	
Quality Requirements	None	
Data, Functions	System functions: take on request, show request details, add information to description field . . .	
Post conditions	The request is closed in the system.	
Included UCs	Clarify request; Handle request	

etc.) and one for the system actions (S1, S2, etc.). Variants are shown right after the related step. There are many variants, if-statements, included use cases, rules and exceptions. Special notation is used to show whether a step is optional, and whether it terminates the use case if done. The description is a kind of program that specifies the possible sequences.

The optional steps may be carried out in any order, which gives the user much freedom to choose the sequence.

Some of A's use cases involve several users and describe a kind of dialog between them. As an example, use case *Clarify Request* describes that a supporter can require more information from the IT user, what the user does, and how a (new) supporter handles it.

Expert A's use cases are very different from H's. As an example, H's entire use case 4 (Transfer request) is just step A3 in A's *Handle request*. While an H use case shows a tiny part of the dialog, an A use case covers a more coherent period.

Supporter assessment: Karin (the senior stakeholder) couldn't understand the program-like details and ignored them. She found the rest okay to read and used it as a checklist. She identified many lines as requirements that were *met* or *want-to-have*. She also noticed some missing or wrong requirements. Morten (the programmer) found the reply very hard to read and spent a lot of time checking the program-like parts. *If I had been asked to read this first, I wouldn't have done it.* He found many parts wrong or dubious. He concluded that the use cases were useless for checking against a new system, since it might well work in some other way.

Task descriptions (Lauesen): Expert L's reply consists of eight task descriptions. Fig. 4 shows one of them in detail (Handle *a request in second line*). It describes what a second-line supporter can do about a request from the moment he looks at it and until he cannot do more about it right now (a *closed task*). He has many options, e.g. contact the user for more information, move to the problem location, order something from an external supplier - or combinations of these.

At first glance, task descriptions look like use cases, but there are several significant differences:

1. The task steps in the left-hand column specify what user and computer do together without specifying who does what. Early on you don't know exactly who does what, and in this way you avoid inventing a dialog. Variants of the task step are shown right after the step, e.g. *1a* about being notified by email.
2. You can specify problems in the way things are done today, e.g. *1p* about spotting the important requests. You don't have to specify a solution.
3. The requirements are that the system must support the tasks and remedy the problems as far as possible. You can compare systems by assessing how well they do this.
4. In the right-hand column you may initially write examples of solutions, later notes on how a potential system supports the step. The left-hand side is relatively stable, but the right-hand side is not. Sometimes the system can carry out the entire step alone, for instance if it automatically records the IT user's name and email based on the phone number he calls from.
5. The steps may be carried out in almost any order. Most of them are optional and often repeatable. The user decides what to do. The steps are numbered for refer-

ence purposes only. (Cockburn and others recommend this too, but it is hard to realize with use cases, because dialogs are a step-by-step sequence.)

Problem requirements. In the example there are four problems in the way things are done today. Some of them, for instance 7p, are very important but not easy to solve.

When a step has an example solution, the solution is not a requirement. Other solutions are possible. Steps without a solution may have an obvious one that the analyst didn't care to write (e.g. step 6) - or he cannot imagine a solution (e.g. 7p). Both are okay.

Lauesen's task list is quite different from expert H's use case list. As an example, H's use case 3, 4 and 5 are steps in Lauesen's *Handle a Request in Second Line*. The relationship to A's use cases is more complex. An A use case may go across several users and a large time span. In contrast a task should cover what one user does with-

Figure 4. Expert L (Lauesen): Tasks - no dialog. Problems as requirements.

C4. Handle a request in second line

Start: The supporter gets an email about a request or looks for pending requests.
End: The supporter cannot do more about the request right now.

Subtasks and variants:		Example solutions:
1	Look at open second-line requests from time to time, or when finished doing something else.	
1p	**Problem:** In busy periods it is hard to spot the important and urgent requests.	Can restrict the list to relevant requests. Can sort according to reminder time, priority, etc.
1a	Receive email notification about a new request	
2	Maybe contact the user or receiver to obtain more information.	
3	Maybe solve the problem by moving to the problem location.	
4	Maybe work for some time on the problem. Inform others that they don't have to look at it.	Put the request in state *taken*.
5	Maybe order something from an external supplier and park the request.	The system warns if no reminder time has been set.
6	In case of a reminder, contact the supplier and set a new reminder time.	
6p	**Problem:** The user doesn't know about the delay.	The system sends a mail when the reminder time is changed.
7	Maybe close the case.	The system warns if the cause hasn't been set.
7p	**Problem:** To gather statistics, a cause should be specified, but this is difficult and cumbersome today.	
7q	**Problem:** The user isn't informed when the request is closed.	The system sends a mail when the request is closed. The supporter has the possibility to write an explanation in the mail.
8	Maybe leave the request in the "in-basket" or transfer it to someone else.	

p, q: Problems today
a, b: Variants of the subtask

User and computer together

Maybe: The user decides

out essential interruptions from Start (trigger) to End (done for now). This is the *task closure* principle.

Supporter assessment: The senior supporter was excited about this reply: *This is so clear and reflects our situation so well. It is much easier to read than the first one I got. I hope I don't step on someone's toes by saying this.* Although she correctly understood the left-hand side, she tended to consider the example solution a promise for how to do it. Morten read this reply as the first one. He was excited about the start and end clauses because they reflected the real work. He was puzzled about the distinction between first and second line, because the hotline didn't operate in that way when he worked there. Initially he hadn't marked the problems as requirements, but later included them. Both supporters marked most of the steps as requirements that were *met* or *want-to-have*. They didn't notice any missing or wrong requirements.

We have seen also in other cases that the two-column principle and the problem requirements are not fully intuitive. However, once reminded of the principle, readers understand the requirements correctly. (The reply started with a six-line explanation of the principles, but the supporters didn't notice it.)

The senior supporter decided to evaluate the system they intended to buy by means of the task descriptions (she still didn't know whom the author was). She did this on her own a few days later, and concluded that most requirements were met - also those that earlier were *want-to-have*. She could also explain the way the new system solved the *want-to-have* problems.

Task description: Expert K's reply consists of four tasks, very similar to Lauesen's, but K hasn't covered statistics and maintenance of basic data. For space reasons we don't show a detailed task. In general all the task replies are rather similar. We chose K's reply because the author had no supporter experience, had experience with tasks in real cases, and followed the principle of task closure.

Supporter assessment: The senior supporter read this reply a few months after reading the first three replies. She found also this reply easy to read. She had marked eight of the requirements with a smiley and explained that these requirements showed that the author had found the sore spots in a hotline. (Seven of these requirements were problem requirements.) She had marked all the requirements as *met* in the new system, but explained that they currently worked on improving the solution to one of the problems (quick recording of problems solved on the spot). She had noticed that requirements for statistics were missing.

Morten also read this a few months later. He was asked to compare the reply against the old system, since he didn't know the new one. He found the tasks easy to read and noticed six requirements that were not met in the old system (five of them problem requirements). He didn't notice that requirements for statistics were missing.

He also made comments that showed that the task principles had to be reinforced. He surprisingly suggested that the step *estimate solution time* should be deleted - otherwise supporters would be annoyed at having to make an estimate. He forgot that all task steps are optional, and you don't have to carry out an optional step. In a few places he complained about missing or bad *solutions*. He forgot that the solution side is just examples. The supplier should give the real solution.

5 Completeness - Dealing with Existing Problems

Requirements are complete when they cover all the customer's needs. Completeness of the ordinary requirements varied within both groups of replies due to participant's experience, time spent, etc. However, the two groups handled the present problems very differently. If requirements don't cover these problems, the customer may end up getting a new system without noticing that the problems will remain.

The analysis report mentions nine problems in the existing hotline, e.g.: *In busy periods, around 100 requests may be open (unresolved). Then it is hard for the individual supporter to survey the problems he is working on and see which problems are most urgent.*

In principle requirements can deal with such a problem in three ways:

1. Specify a solution to the problem.
2. Ignore the problem.
3. Specify the problem and require a solution (a *problem requirement*).

Problem requirements are unusual in traditional requirements, but are used extensively in tasks. The use case replies deal with problems by either specifying a solution or ignoring the problem.

All the use case replies ignored the problem with the busy periods. Expert A confirmed our suspicion that it was because they couldn't see a solution.

Expert A explained: *We do not think that this can be solved by the system. The system gives all the important information (e.g. date placed) for the user to decide.*

With tasks you can just record the problem, for instance as in Fig. 4, 1p. Lauesen's experience from many kinds of projects is that recording the problem helps finding a solution later, for instance the one outlined in Fig. 4, 1p. Even if the analyst cannot imagine a solution, a supplier may have one. As an example, we considered problem 7p (specifying the cause of requests) hard to solve until we saw a product with a short experience-based list of the most common causes. They covered around 90% of the help requests. The customer could gradually add more causes.

The analysis report mentions nine problems. For each reply we checked which problems it covered, either as a problem requirement or as a solution. Fig. 5 shows the results. The problem above is recorded as problem A. None of the use case replies deal with it. Problems that have an easy solution, for instance problem I, are covered by most replies.

Tasks cover these nine problems significantly better than use-cases do:

- Number of problems covered by a use case reply: 3.8 ± 1.2
- Number of problems covered by a task reply: 6.5 ± 1.9

The difference is significant on the 1% level for ANOVA (p=0.6%) as well as for a t-test with unequal variances (p=0.5%).

The standard deviations are due to differences in expertise, time spent, simple mistakes, etc. Not surprisingly, the deviation for task replies is larger because some task participants had little task experience (N and O).

Maiden & Ncube [16] observed that when comparing COTS products, all products meet the trivial requirements. Selection must be based on the more unusual requirements. In the hotline case, the problem requirements are the ones that will make a difference to the customer, while the ordinary requirements will be met by most systems.

6 Too Restrictive Requirements

In the hotline case the purpose is not to develop a new system, but to expand the existing one or acquire a new one (probably COTS-based). With the task technique stakeholders check how well the system supports each task step and each problem. In order to compare systems, they check each of them in the same way.

We couldn't see how the use cases handled this situation. The use cases specify a dialog in more or less detail, and it seems meaningless to compare this dialog with a different dialog used by an existing system. The requirements arbitrarily restrict the

Figure 5. Problem coverage: 1 = fully covered, 0.5 = partly covered.

Use-case based:	A: Hard to spot important requests	B: Difficult to specify a cause. May change later	C: User: When can I expect a reply?	D: Forgets to transfer requests when leaving	E: Cumbersome to record on-the-spot solutions	F: User: When is it done?	G: Nobody left on 1st line	H: Reminder: Warn about overdue requests	I: Today it is hard to record additional comments	**Total**
Expert A				1	1	1	1	0.5	1	5.5
Expert B		0.5	1		1	1		0.5	1	5
Expert C				0.5		1	1	0.5	1	4
Expert D				1			1	1	1	4
Expert E					1		1	1	1	4
Expert F						0.5	1	1	1	3.5
Expert G			0.5					0.5	1	2
Expert H						0.5		0.5	1	2
Total UC	0	0.5	1.5	2.5	3	4	5	5.5	8	
Task-based:										
Expert I	1	1	1	1	1	1	1	0.5	1	8.5
Expert J	1	1	1	1	1	1	1		1	8
Expert K	1	0.5	1	1	1	1	1	1		7.5
Expert L	1	1	1	1	1	1		1		7
Expert M	1	1	0.5		0.5	1	1		1	6
Expert N	1	1			0.5	1			1	4.5
Expert O					0.5	1	1		1	3.5
Total tasks	6	5.5	4.5	4	5.5	7	5	2.5	5	

solution space. So we asked participants how their use cases could be used for comparison with an existing system. Here are some of the replies:

Expert F: *We will* make *use of something that we call a decision matrix. We have prepared a sample for your reference where we are comparing the present Hotline system with the system that we propose…*

Their matrix shows nine features to compare, e.g. *Reminder management* and *Automatic request state management*. They have no direct relation to the use cases. F explained that their use cases specified a new system to be developed from scratch.

Expert H: *Use Cases are an optimal source for defining Test Cases. So running these Test Cases against different proposals and the existing system, it will be clear which systems fulfill most of the functional requirements. The tests can be run "on paper" since many solutions have not been developed yet.*

This seems a good idea. The only problem is at what detail you define these test cases. Assume that you use the use cases directly as test scripts. You would then test *Transfer request* (Fig. 2) by trying to select a request from the list, checking that the system shows request details and a list of hotline employees. But what about a system where you don't need to see the details, but transfer the request directly from the list? You would conclude that the system doesn't meet the requirements (actually it might be more convenient). One of the supporters (Morten) actually tried to verify Expert A's program-like use cases in this way, but became so confused that he gave up.

Hopefully, the testers have domain insight so that they can abstract from the details of the use case and make the right conclusion. If so, most of this use case is superfluous. You could omit everything except the heading *Transfer request* and simply test that the system can transfer a request, and make a note about how easy it is to do so. The other supporter (Karin) actually verified all use cases in this way.

Expert A: *Our requirements only describe the to-be system. They cannot be used to compare the other systems with the current (open source) system. However, they could be used to see whether the supplier's system (if not available, the description of the system) meets our requirements.*

Expert A (another team member): *I think the purpose of your [Lauesen's] specification is quite different from ours. We want to provide a specification that describes the solution to the problems on a high level. For the purpose of choosing between solutions your specification is much better, but this was not so prominent in the experiment description that we considered it the major context.*

The other use-case replies follow the same lines. They describe a future solution in detail.

Why did all the use case authors describe a future solution and later realize that it wasn't useful as requirements in this case? The analysis report said that the customer wanted to modify the existing system or buy a new one - not build a new one. We believe that use case principles and current practice are the causes:

1. Use case principles force you to design a dialog at a very early stage. In this way you design key parts of the solution rather than specifying the needs.
2. Use cases are so widely used that nobody questions their usefulness.
3. Few analysts know alternative requirements that are verifiable, not solutions, and reflect the context of use.

7 Wrong requirements

The supporters and we noted several wrong requirements in the use cases. As an example, Expert C mentions these two business rules:

R1. Only problems with high priority may be requested via phone or in person.

R2. For statistical purpose it is not allowed to create a request for more than one problem.

None of these rules are justified in the analysis report, and it would be harmful to enforce them. Should hotline reject a user request if it contains more than one problem? The hotline would surely get a bad reputation.

We believe that use case theory and templates cause these mistakes. Many textbooks on use cases emphasize rules, preconditions, etc. and their templates provide fields for it. Most replies used such a template, and as a result, the authors were tempted to invent some rules, etc. Often these rules were unnecessary or even wrong.

In order to avoid this temptation, tasks do not have fields for preconditions or rules. When such a rule is necessary to deal with a customer need, it can be specified as a task step, e.g. *check that the request has a high priority,* as a constraint in the data model, or in other sections of the requirements [13].

8 Conclusion

In this study we compare real-life use cases against the related technique, task description. We deal only with use cases that specify the interaction between a human user and the system. We do not claim that the findings can be generalized to other kinds of use cases, for instance system-to-system use cases.

The study shows that with use cases the customer's present problems disappear unless the analyst can see a solution to the problem. The consequence is that when the customer looks for a new system, he will not take into account how well the new system deals with the problems. Even if the analyst has specified a solution, a better solution may not get the merit it deserves because the corresponding problem isn't visible in the use cases.

Task descriptions avoid this by allowing the analyst to state a problem as one of the "steps", with the implicit requirement that a solution is wanted (a *problem requirement*). Example solutions may be stated, but they are just examples - not requirements. In practice, stakeholders need some guidance to understand this principle.

The study also shows that use cases in practice produce too restrictive requirements for two reasons: (1) They force the analyst to design a dialog at a very early stage, in this way designing a solution rather than specifying the needs. Often the dialog would be very inconvenient if implemented as described. (2) Many use case templates provide fields for rules, preconditions, etc. and these fields encourage analysts to invent rules, etc. Often they don't reflect a customer need and may even be harmful.

Task descriptions don't specify a dialog but only what user and system need to do together. The supplier defines the solution and the dialog, and stakeholders can compare it against the task steps to be supported. Tasks don't tempt the analyst with fields for rules, etc. When rules are needed, the analyst must specify them as separate task steps or in other sections of the requirements.

References

1. Achour, C. B., Rolland C., Maiden N. A. M., Souveyet, C.: Guiding Use Case Authoring: Results of an Empirical Study. Proceedings of the 4th IEEE International Symposium (1999)
2. Armour, F., Miller, G.: Advanced Use Case Modeling, Addison-Wesley (2001)
3. Cockburn, A.: Writing Effective Use Cases, Addison-Wesley (2000)
4. Constantine, L. L., Lockwood, L. A. D: Software for Use: A practical guide to the Models and Methods of Usage-Centered Design, Addison-Wesley, New York (1999)
5. Cox, K., Phalp, K.: "Replicating the CREWS Use Case Authoring Guidelines", Empirical Software Engineering Journal, Vol. 5, No. 3, pp. 245-268 (2000)
6. IEEE Recommended Practice for Software Requirements Specification, ANSI/IEEE Std. 830 (1998)
7. Jacobson, I., Christerson, M., Johnsson, P., Övergaard, G.: Object-Oriented Software Engineering - a use case driven approach. Addison-Wesley (1992)
8. Jacobson, I.: Use Cases: Yesterday, Today, and Tomorrow, IBM Technical Library (2003)
9. Kulak, D., Guiney, E.: Use Cases: Requirements in Context, Addison-Wesley (2000)
10. Lauesen, S.: Software requirements - styles and techniques. Addison-Wesley (2002)
11. Lauesen, S.: Task Descriptions as Functional Requirements. IEEE Software, March/April, pp. 58-65 (2003)
12. Lauesen, S.: User interface design - a software engineering perspective. Addison-Wesley (2005)
13. Lauesen, S.: Guide to Requirements SL-07 - Template with Examples, 2007, ISBN: 978-87-992344-0-0. Also on: www.itu.dk/people/slauesen/SorenReqs.html#SL-07.
14. Lauesen, S., Kuhail, M. A.: The use case experiment and the replies (2009): www.itu.dk/people/slauesen/
15. Lilly S.: Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases, IEEE Computer Society, Washington, DC,US (1999)
16. Maiden, N. A., Ncube, C.: Acquiring COTS software selection requirements. IEEE Software, March/April, pp. 46-56 (1998)
17. Rosenberg, D., Scott K.: Top Ten Use Case Mistakes, Software Development (2001): http://www.drdobbs.com/184414701
18. Sigurðardóttir, Hrönn Kold, project manager for the Electronic Health Record system at the Capital Hospital Association (H:S): Draft of PhD thesis (2010)
19. Wirfs-Brock, R.: Designing Scenarios: Making the Case for a Use Case Framework, Smalltalk Report, Nov/Dec (1993). Also: http://www.wirfs-brock.com/PDFs/Designing%20Scenarios.pdf