In: Proceedings of REFSQ'99, Heidelberg, June 1999-06-24

Use Cases in a COTS Tender

Soren Lauesen & Marianne Mathiassen

slauesen@cbs.dk, marianne.mathiassen@get2net.dk
Copenhagen Business School, Howitzvej 60
DK-2000 Frederiksberg

Abstract. When a customer buys a large standard system through a tender process, he has little influence on the detailed functionality. This calls for functional requirements on a high level. In this study of a large payroll and roster system, we have tried use cases as the requirements. They turn out to give both customer and supplier a better understanding of the demands, they don't favour one supplier over others, and they help elicit important requirements. To replace traditional requirements, they have to be supplemented with data models. Still some requirements are difficult to cover through use cases, e.g. those specifying complex screens for monitoring purposes.

1. Introduction

COTS systems (Commercial Off The Shelf) cover more and more application areas, and in many cases the supplier can modify and extend his COTS system to suit even larger areas of use. When several competing COTS systems are available for the same application area, it makes sense to purchase the system through a tender pro??cess. Because of EU acquisition rules, this is mandatory in public organisations in the EU.

From a requirements point of view, the situation is difficult. Imagine a specific feature required by the customer, e.g. *the system shall accept data x and as a result show data y*. Some suppliers may provide the feature as a standard part of their product, others may provide it as a tailor-made extension, and still others may not have that feature, but suggest that the customer use two other features to obtain essentially the same result. What to write in the tender requirements?

Obviously, detailed features such as screen pictures should not be specified. They would favour some suppliers in a rather arbitrary way. High-level features seem to be the answer, but it is difficult to make them truly supplier-independent.

An alternative approach would be to specify the user tasks that have to be supported, and then invite suppliers to specify how they can support these tasks. This is the approach we have taken in this case study. We have chosen to specify the user tasks as *use cases*, since this technique is gaining wide acceptance.

The purpose of our case study is to develop a version of use cases which could replace major parts of feature-based requirements, and give advantages such as supplier-independence, increased understandability, and better coverage of demands. Our case project is a tender for a large payroll and roster planning system to be used in public hospitals, rest homes, etc.

2. Use Cases

The use case concept was introduced by Jacobson et al. [1994]. Soon it diversified into many forms. In its basic version, a use case is an interaction between a user (the actor) and the computer system. You can also say that it describes a task to be performed by human and computer together. The interaction can be described on different levels of granularity. Many developers use low-level use cases, such as this example from roster planning:

Low-level use case:
Record vacation for staff:
Step 1: User enters staff id.
Step 2: User enters vacation period.
Step 3: Computer indicates whether vacation conflicts with regulations.

In case of a COTS system, this is of no use as a requirement since many other procedures are possible for recording staff leave. Researchers rightly question the general usefulness of this level for requirements. Essentially, it is a premature design of the dialog.

Our prime source is Cockburn's use cases [Cockburn, 1997], because they are on a high level, and they have a goal, i.e. something to be achieved for the use case to terminate and the user to feel that he has achieved something useful. Here is an example of a high-level use case from roster planning:

High-level use case:
Allocate duties
Goal: Staff all duties with competent people. Ensure that regulations are fulfilled, leave respected, . . .
Step 1: Initialise new roster period with standard allocations
Step 2: Record leave for staff
Step 3: Allocate staff for unstaffed duties
Step 4: Send roster for internal review
. . .

This version is quite supplier-independent, particularly if we assume that the steps are not a strict sequence, but may be interleaved to some extent. The same steps have to be performed whatever COTS we acquire, although the way they are performed may vary a lot.

Some researchers claim that high-level use cases should not be used as requirements. They may be very useful in analysis, but "real" requirements have to be extracted from them [Kovitz, 1999; Meyer, 1997; Graham, 1998; Sommerville & Sawyer, 1997]. A "real" requirement would be a product feature such as *the computer shall record staff leave and show whether it conflicts with regulations*.

Other researchers seem to be happy with use cases as requirements, e.g. [Larman, 1998; Cockburn, 1997; Schneider & Winters, 1998].

Our view is that it is difficult to understand the required product features without understanding the context (the use case) in which they will be used. The context is also important for a good implementation where users don't have to flip between a lot of screens to accomplish a single use case. Only knowledge of the use case will prevent the developer from making one screen for the staff leave feature, another screen for allocation, etc.

In: Proceedings of REFSQ'99, Heidelberg, June 1999-06-24

So we suggest that use cases appear in the written spec, either as requirements or as justification of required features. In this case study we explore the possibility of using them as requirements directly.

3. Hospital Payroll and Roster

In September 1998 we established contact to the IT department of a Danish local government (county or *amt*). They were in the process of acquiring a new payroll and personnel system to support several hospitals, rest homes, high schools, etc. The old system was largely batch-oriented and it didn't support critical tasks like roster planning. Several local governments were running similar processes in that period. Our contact had acquired a few large IT systems through tenders, but with problems of many kinds, e.g. that the supplier failed to deliver. Usually, it is very difficult to get inside information about such processes; politics are involved and good relations to the suppliers have to be maintained. Probably because one of us (Lauesen) had studied several projects and related system success to requirements, we got permission to see the requirements specification.

In our first review of the spec, we noticed that it was better than average in several areas, for instance security, interoperability, and staff education. In other areas, we saw weaknesses, for instance a mixture of project goals, wishes, and required system features. It was difficult to see whether the high-level project goals were reflected in the required features, what the features really were, why they were required, and whether they were reasonably complete. Still, the weak parts compared well with many other specs we had seen.

During a meeting with the IT department, the staff confirmed our observations and explained that the good parts had been made by the IT department based on their experience with other tenders, while the more problematic parts essentially had been made by user departments (e.g. hospital administration). The IT-people had the same problems with these spec parts as we had. They were not sure the requirements were adequate, but hoped the user department had got them right.

During the second half of our meeting, we showed examples of different ways to express functional requirements and usability requirements. The IT people immediately found the use case concept interesting. (Our use case example was based on Cockburn's version with goals and high-level textual descriptions.) They realised that it was a way to specify the tasks to be supported rather than the necessary system features. In principle, it would solve two problems for them:

- Make the spec supplier-independent. The features in the present spec tended to favour the present supplier, since they matched his product quite well. With the use case approach, no specific features were required.
- Allow the IT department to assess the spec and discuss solutions with suppliers. The use cases allowed them to understand what was necessary.

Use cases would probably solve other problems too, for instance how to specify usability requirements in a measurable way, how to evaluate a demo of the system, and how to verify requirements at deployment time. However, these issues seemed to be of minor importance to IT staff at that moment.

The IT people raised an important issue. With the existing spec, the supplier's bid stated the features he could supply as a standard, features supplied as an extension, and features he couldn't supply. With use cases, what would the bid contain? Our

Project goals for personnel department:

A Automate some user tasks (supported by 1.2, 3.2, 3.3)
B Remove error sources (supported by 3.2, 3.3)
C Improve correctness of data (3.2, 3.3)
D Observe deadlines, particularly the 120 day rule (1.2, 3.2, 3.3, 3.4)
E Less trivial work and less stress (3.2, 3.3, 3.4)

Project goals for hospital department:

F Reduce over- and under-time payment (supported by 1.1)
G Faster roster planning (supported by 1.1)
H Improve roster quality (right person on right duty, observe regulations and leave days, etc.) (supported by 1.1, 2.2)

Use cases. User: Planner in department

1.1 Allocate duties for next period (faster, better quality, reduce pay)
1.2 Report actual work hours to personnel department (automate)

Use cases. User: Staff in department

2.1 Record actual work hours when deviating from plan
2.2 Find replacement in case of illness (better quality)
2.3 Swab duties between two staff members

Use cases. User: Personnel department

3.1 Record employment and dismissal
3.2 Enter data about actual work hours for payroll calculations, flex status, etc. (automate)
3.3 Record illness. Start dismissal action in case of 120 days of illness. (deadline)
3.4 Supervise that reporting from department is on time (hardly done to-day)

Fig. 1. Project goals and use cases for the roster part of the system.

answer was a general statement that the supplier had to specify how he would support the use cases. But what it meant in practice, we couldn't say at that time.

We suggested that we tried to develop use cases for a non-trivial part of the system, roster planning and work registration. We would evaluate the use cases with user department, IT department, and potential suppliers.

4. Requirements Elicitation

Since the IT department had only a weak understanding of the user tasks, we met with an IT person and an expert user from one of the hospitals. They had both been closely involved in the spec.

The meeting was four fascinating hours. The expert user understood the use case principle immediately, based on an example. She outlined the high-level purposes of the project, and then we together outlined the essential use cases (9 use cases sufficed). Finally, we wrapped up by rephrasing the project goals and discussing whether they were reflected in the use cases. Back home we summarised the result as shown in Fig. 1. It lists the project goals and the use cases. It also shows how they relate together, something that turned out to be surprisingly easy.

(The "120 day rule" is mentioned a few places. It is a Danish union agreement that when an employee has been ill for 120 days within a year, the employer is entitled to dismiss the employee with short notice, assuming that he does so the last of the 120 days. If the employer doesn't observe the deadline, he cannot dismiss the employee immediately. The problem is that decentral departments know about the illness, but the central personnel department has to start the dismissal action. The proper exchange of information is often forgotten.)

Finding use cases and project goals in four hours was of course only possible because user and IT person had been closely involved in the old spec. Further, the two researchers had studied the existing spec closely. An interesting observation was that the IT person was amazed by the level of understanding she acquired during the process. As an example, she observed that the system they had contracted to get, probably would lack a very essential feature. Nobody in IT had realised that, since they didn't understand the user tasks. She also explained that her impression was that suppliers often signed a contract without really understanding the requirements. Most likely, suppliers would also understand better.

During the meeting, we had noticed several methodological problems with use cases:

1. Do we describe the present tasks, the imagined future tasks, or essential tasks (technology-independent)?
2. How do we relate overall project goals to requirements?
3. How do we specify the information aspect, e.g. the necessary information to carry out a task?
4. What is the overlap between the use-case-based spec and the traditional spec? What to preserve from the traditional spec?
5. How do we specify critical timing problems?
6. How do we specify data transfer between different departments (different use cases)?

Below, we will describe our solution to the first four of these problems and give excerpts from the final use cases.

5. Specify Old, New, or Essential Tasks?

Before object-orientation and use cases became fashionable, structured analysis and design (SA/SD) was the fashion [De Marco, 1979]. This method used concepts such as events, processes, and dataflow diagrams. We will for a moment recast the use case steps into these traditional terms.

Fig. 2 shows a dataflow diagram of step 3 from the use case "allocate duties". The bubble is a process that takes input data and produces output data as shown by the arrows. This bubble is the use case step. It takes information about regulations, personnel, existing duties and assignments, tacit knowledge about staff preferences, and produces new assignments.

In contrast to the use case format, the input and output are clearly visible in the dataflow diagram. In full SA/SD the detailed data contents of each arrow is shown in a separate listing.

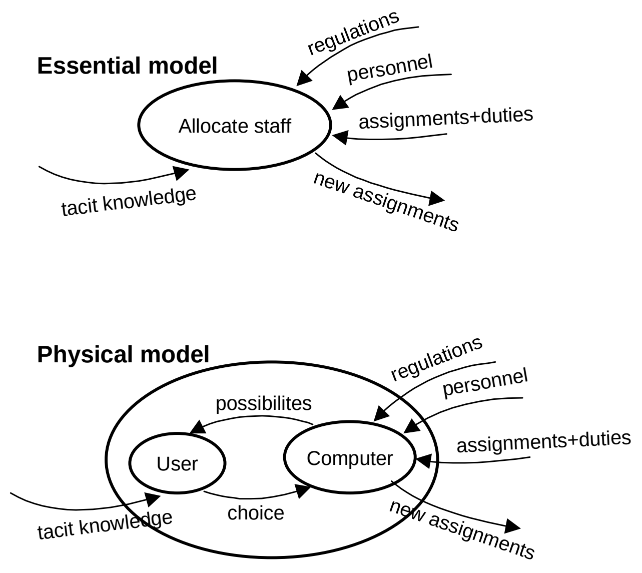

Fig. 2. A use case step shown as two dataflow diagrams. The essential model is implementation independent. The physical model splits the activity into a user process and a computer process.

In our example, the bubble corresponds to the joint action of user and computer. The bubble is technology independent in the sense that the input, and output will be the same no matter which technology we use inside the bubble. Constantine, Yourdon, and De Marco call the diagram an *essential model* [Yourdon, 1989] or sometimes a logical model.

When we ask for an IT system to support the task, we divide the process into two sub-processes, one to be performed by the user and another to be performed by the computer. Constantine, Yourdon, and De Marco call it a *physical model* of the process. The figure shows a version where we get good support from the computer. The computer does the hard work of balancing regulations, personnel, etc. and just presents a prioritised list of possibilities to the user. The user makes a choice based on tacit knowledge of staff preferences.

There are many ways to divide the bubble into human and computer part. Some suppliers suggest one way, some another way. The present, old way of performing the task is yet another way of dividing the bubble into a human part and a machine part.

Based on this analysis, we can ask the question: What do we describe in the text-based form of the use case: the essential model? what the user does now? what the user will do in the new system? what the computer will do? Let us look at some possibilities for the allocation step in light of the COTS situation:

Use case with essential actions:
Step 3, Allocate staff: Select staff for unstaffed duties.

Comment: This version describes what user and computer perform together. Note the imperative style which hides who does what. This is a good description of what to accomplish (the goal), but what is the requirement? The step can in principle be done manually, but that is hardly sufficient. Is the requirement to support the step? How? The supplier might specify how he supports the step, but what is good enough?

Use case with user action in present system:
Step 3, Allocate staff: Planner tries various combinations using paper and pencil.
Comment: This version gives background information, but what is the requirement? Could, however, illustrate what the present problems are. That would help the supplier specify how he solves the problem, particularly if the problem calls for an addition to the COTS system.

Use case with user and computer action in new system:
Step 3, Allocate staff: Computer shows staffing proposals. User selects actual staff.
Comment: This is a natural explanation of what the user tells the computer and the computer tells the user. It is close to a traditional requirement, because we specify the services to be provided by the computer. Unfortunately, it is difficult to use in a COTS situation because some COTS systems may provide the service in one way, some in another. Experience shows that it is hard to imagine all the ways it could be done. Specifying one of the ways, would favour some suppliers. So at least the description would have to be on a rather high level.

We experimented with many combinations of these possibilities. Inspired by the matrix of old and new tasks [Checkland and Scholes, 1990, p42] and Karlheinz Kautz's practical demonstration of the idea (private communication), we tried to show three versions in a single matrix: the essential model, the present actions, and the new actions. That turned out to be a lot of writing and it was difficult to get an overview of the use case. In particular, the difference between present user actions and new user actions was often trivial. Example:

Present user action: User records staff vacation on paper.
New user action: User enters staff vacation.

We ended up with the proposal illustrated in Fig. 3. Each step covers:

Essential action. Often given just as a one-line imperative - the name of the step. In some cases more information is needed.

Present problems, if any. We state part of the present user action only if it is problematic.

Example of solution (new action). Described on a rather high level. It is not a requirement, but an example of how the new system could support the task. In many cases we don't specify the "solution" because it is trivial - the user just enters some trivial information.

The supplier's solution. This doesn't have a separate field. The supplier gets the spec as a file and modifies the example in such a way that the customer can see the essential aspects of the solution. The supplier states whether the solution consists of standard features, small modifications, or larger extensions. He may state several solutions, for instance more or less standard.

Why do we write an example of a solution? Logically, this is not necessary, because the supplier has to specify his solution. The example is not the requirement. Actually, the example helps the supplier understand the requirement and formulate his solution in terms understandable to the customer. It also helps the customer state his expectations and the possible benefits.

Fig. 3 shows the result of this approach for the most complex use case, Allocate duties. The other eight use cases are much simpler. Some of them may be completely automated so that no human action remains (use cases 1.2 and 3.2). One use case is not really carried out to-day, but has to be in the future (use case 3.4).

You might wonder why the regulations are not described in detail. The developer has to know them in order to implement the system. The reason is that the regulations are extremely complex and neither IT department, nor personnel department knows them fully. The customer explicitly stated that they only wanted bids from companies that knew the regulations and had built them into their product.

6. Project Goals versus Requirements

Above we have intentionally used the term "project goal" rather than "system goal". The two things are quite different in our context. *System goals* are those we talk about in goal-oriented use cases. They are the goals to be accomplished when a use case is carried out. *Project goals* are the reasons for investing in a new system. Here is an example of the difference:

Use case goal: Allocate staff so that all duties are staffed.

Project goals: Save manual work, improve roster quality, and reduce overtime payment.

A frequent problem with requirements is that they don't reflect the project goals. The result is often that the new system fulfils the requirements, but not the expectations. The structure of the requirements has significant influence on the ease of verifying the project goals.

With traditional requirements, it is quite hard to ensure that project goals are met through the many features specified. With the kind of use cases we apply, it is easy. The outline in Fig. 1 shows how it can be done. Goal A, for instance, is satisfied by automating use cases 1.2, 3.2, and 3.3. If a supplier fails to automate one of these tasks, it has serious consequences for the project goals. (In the roster case, the new system failed to automate use case 1.2).

In our detailed version of the use cases (Fig. 3), such a goal relation turns up as a "problem" in one or more of the steps. In the proposal made by the supplier, we can easily see how well he solves the problem, we can check it out in the demonstration he most likely makes, and we can verify it at delivery time.

7. The Information Aspect

When we compare use cases with SA/SD, the use cases correspond to the dataflow diagrams; they show the processes carried out. However, SA/SD takes great effort in specifying the data coming in and out of each process. Further, with modern versions of SA/SD, you develop an information model (data model, e.g. an entity-relation model) in parallel with the dataflow diagram. The information model is supposed to provide the data flowing in and out of the process bubbles. Good CASE tools are able to check that this is actually the case.

In the object-oriented world, various class diagrams replace information models. Unfortunately, neither entity-relation models, nor class models link well to the use cases. There are several reasons for that, one being that the use cases don't explicitly mention the data. In the use case in Fig. 3, for instance, you may notice statements like "system generates a roster". What exactly does that mean? Which fields are involved? What is shown to the user?

As another example, what do we mean by "allocate staff"? Which staff? Staff in the department? Could the same person belong to several departments?

We feel it is important to specify the information needs for each step, rather than leaving it to the supplier/developer to decide what is needed. On the other hand, it would clutter the picture if we tried to fully specify information needs inside the use case itself. One sign that information has to be specified is that a supplier with expert knowledge in payroll and union regulations, but no experience in hospital rosters couldn't develop a reasonable roster system based on the use cases as they are. (The existing spec wouldn't help either.)

We experiment with specifying the information needs as "virtual screens" to be read in parallel with the use cases, but it gets close to designing the actual interface, what is premature in a COTS tender [Lauesen et al., 1994].

We feel more confident about suggesting the use of an entity-relation model as a supplement to the use cases. It is an extremely well proven way of analysing and specifying the information to be kept in a database. For instance, it would immediately draw attention to the issue of staff belonging to several departments. It would also clarify what a roster is and what information we could display about a roster.

Unfortunately, a data model is a specialist affair and non-technical people cannot read it. But in our COTS case, the IT specialists could make the data model and use it for two things:

1. Specify what data the system should be able to hold.
2. Check that the use cases cover all data (CRUD check).

The CRUD check involves looking at each piece of data in the model and ask: Who Creates this data? Who Reads it? Who Updates it? Who Deletes it? Are there important use cases where this has to be done? Usually, several important use cases can be derived in this way.

In conclusion, we suggest to develop a data model (or a class model with precise associations) in parallel with the use cases. The data model should be part of the requirements as the specification of data to be maintained by the system.

8. Match with Existing Spec

The first step in checking that our use case approach would be an improvement, is to match it against the existing spec. Which parts of the old spec were not covered by the use cases and vice versa.

Use case:	**1.1 Allocate duties**	
Goal:	Staff all duties with competent people. Ensure that regulations are fulfilled and leave respected. Ensure low cost, e.g. avoid overtime and unused staff.	
Frequency:	Once every two weeks. In some departments more seldom. A round of planning is a complex process that may be done over several days.	
Critical:	Vacation periods where much regular staff is off. Vacations are allocated several months in advance.	
Steps:		**Example of solution:**
1	**Initialize new roster period** Typically, planning done in week x covers week x+5 to x+8 for nurses, week x+7 to x+32 for surgeons.	System generates a roster (duties and assignments) for the new period based on a standard roster template. User can modify the roster as well as the roster template.
2	**Record staff leave** There are two kinds of leave: 1) promised leave, e.g. vacation and birth leave; 2) wished leave, e.g. flex time. Promised leave has to be respected by the planner, while wished leave need not. **Present problems:** Leave requests are kept on a separate kalendar or manual notes, often going months into the future. Requests are sometimes forgotten when the plan for that period is made.	System can record leave one year into the future, although no roster has been made yet for that period. System warns if leave is against regulations. It must be easy to record a long period of leave (several months).
3	**Allocate staff for unstaffed duties.** Staff may be regular or temporary. Ensure level of competence, regulations, leave days, and low cost. **Present problems:** With the manual method, it is difficult to ensure this. Costs are higher than necessary and errors occur.	System shows unstaffed duties and suggestions for staffing. User selects the actual staff. System warns if duties are unstaffed, leave or regulations violated, or cost unnecessary. System supports extensive undo of user actions and allows several temporary versions of a plan.
4	**Send roster for internal review**	A print of the roster is sufficient.
5	**Modify roster**	Steps above suffice
6	**Authorize roster**	User selects one of the temporary rosters as the authorized one for salary and flex time purposes. A print of the roster is made too.
Extensions:		**Example of solution:**
4a	**Staff not yet recorded in the staff file**	User enters preliminary data for new staff. Later, the personnel department makes the final record.
4b	**No staff available with necessary competence** **Present problem:** User doesn't have information about available staff in other departments	System suggests staff from other departments based on their authorized rosters.

Fig. 3. Final use case for "allocate duties". For each step (sub-task) we show the essential action to take place (user and computer together), possible problems in the present way of doing things, and an example of a solution. The supplier is supposed to replace the example with his actual solution.

In: Proceedings of REFSQ’99, Heidelberg, June 1999-06-24
© Soren Lauesen, 1999

8.1. Uncovered Parts of the Old Spec

We reviewed that part of the original spec which dealt with rosters and time registration, to see whether it was covered by the use case model. One might believe that everything would be covered since the researchers had studied the old spec. Actually, we had based the use cases on the elicitation meeting, and details of the old spec were far back in our mind. This also shows in the comparison. Here are our observations of the existing 42 requirements for roster and time registration:

12 requirements were fully covered by the use cases. Two of them were more on the goal level, but covered anyway. Example: *All data involved in time registration shall only be recorded once.* (Covered by use case 1.2, 3.2, and 3.3.)

4 requirements gave additional details to something covered by the use cases. Example: *All regulations, including individual local agreements, shall be handled by the payroll calculation.*

2 requirements would have been covered by an information model. Example: *It shall be possible for an employee to work in several roles and several departments at the same day.*

5 requirements would have been caught if we had made a CRUD check. Example: *If an employee works in more than one department, it must be possible to partition his total pay between the departments.* (We assume that the information model would have shown that cost data related to several departments.)

15 requirements specified output (reports on screen and paper) that wasn't demanded by any of our use cases. Example: *The system shall be able to show a vacation account for each employee.*

3 requirements were strange. We couldn't understand what they meant.

1 requirement specified something that wasn't a true requirement: *The system shall consist of a planning part and a time registration part.*

42 requirements in total

In addition to these functional requirements, the spec contained non-functional requirements in separate chapters. In our case study, we didn’t try to cover them. (A systematic way of deriving usability requirements in a tender process was published by Lauesen [1998]).

The most interesting thing is the 15 output requirements that were not demanded by the use cases. If they are true requirements, there should be use cases that demand them. We know that there are some use cases that we didn't try to cover, partially because the user wasn't too sure about the actual procedures. They involved budgeting and follow up on the budget. If we had covered these use cases, we might also have covered about half of these 15 output requirements.

However, we also believe that there are a lot of use cases about someone wanting to see how this or that totals, what the history of something is, etc. These use cases are probably quite important, but difficult to get hold of since they don't have a clear operational goal. They belong to a broader class of requirements about *getting an overview* of some (complex) data.

To some extent, a CRUD check might hint at a need for seeing some of these data. As an example, a CRUD check should ask *who reads the information about vacation days spent*. The answer would include that both employer and employee might want to see them, and it would be reasonable to show also the total number of vacation days and the number of days left. With more complex data it is not so obvious what to show. In the counts above, we have not assumed that a CRUD check could reveal the complex output requirements.

8.2. New Requirements Through Use Cases

We also reviewed the use cases to see which requirements were new. We counted eight new requirements:

1. Support needed for vacation much later than the current planning horizon.
2. Automation needed for use case 1.2.
3. Need for several temporary versions of a roster and for extensive undo.
4. Need for including people in the roster before they are formally recorded.
5. Need for alarming the personnel department shortly before the end of a 120 day sick leave. (The present spec mentions the problem, but not as a requirement that solves the problem.)
6. Support needed for two people swapping duties.
7. Support needed for an employee turning ill on duty.
8. Need for supervising that roster planning and work time registration is performed.

In addition, the use cases added detail and purpose to several existing requirements. The reason we could come up with the new requirements was that the work with use cases allowed us to imagine the new work procedures with a degree of clarity that traditional requirements didn't allow.

We know that the system, which the customer has contracted to get, does not satisfy requirement 1, 5, and 8 above. If these requirements had been stated and checked at proposal time, the customer might well have chosen another supplier or asked for modifications of the standard system.

9. Review with Suppliers

The next step was to review the use cases with the stakeholders to see advantages and disadvantages of the approach. At the time of writing, we have only made a review with the suppliers. The IT-department and the users have been too busy with other things.

In the Danish market there are only three suppliers of this kind of system. They had all been involved in the case we had studied. We have reviewed the use case approach with each of them.

We conducted the first two reviews in this way: We had a meeting with one or two consultants who had been involved in the bidding process. We briefly told the case story and then explained the example in Fig. 3. Essentially, they understood the idea immediately (within 3 minutes). Next we asked them to study the use cases more closely and tell us about any advantages and disadvantages they could see. This took about 20 minutes. We then switched into a more open discussion of the market situation, customer behaviour, etc. Finally, we asked them to complete the right column of the use case with their proposed solution.

The last review was prepared by e-mail. We had never met the consultants but sent them an e-mail with a short explanation of the idea and the full use cases. On their own initiative, they filled out the right hand column as if it had been a tender process, and brought the result to a meeting with us a few days later. Here we asked for their opinion as in the first two reviews. Essentially they had the same opinion, but were a

In: Proceedings of REFSQ’99, Heidelberg, June 1999-06-24

bit more enthusiastic about the approach. They ended up encouraging us to teach the counties "this new way of writing specs".

Advantages

All three suppliers confirmed our early observations that the approach didn't favour any supplier, and vastly improved understanding of requirements and customer expectations. The whole idea corresponded to what they usually discussed with customers, but never wrote down. The approach would also allow the customer to evaluate the proposals faster and better.

The use cases were a good basis for demonstration of the product and for acceptance test at the end. One supplier commented that the traditional feature-based requirements could be verified more exactly, but with less meaning to the customer.

They all mentioned the need for a process re-engineering perspective in the customer organisation and saw the use cases as a vehicle for that. They also liked the ease with which project goals could be traced to requirements. One consultant was fascinated with the use case goals (Cockburn's idea), saying that this was very essential, but a big problem to most customers.

Problems and issues

Most of the negative comments related to things we didn't expect to handle with the use-case approach. We agreed to most of them.

Security wasn't covered. Relationship between tasks in different departments wasn't visible (we explained that a higher-level use case might help). The approach couldn't be used for negotiation on top management level.

The first consultant felt that the approach couldn't be used with the standard government contract (called K18), but later consultants saw no problem in that area - even if asked explicitly.

All suppliers found that customers would take longer time to make requirements based on use cases. At present large parts of tender specs were reuse from other specs. Customers would need training.

Willing to try?

All suppliers were willing to try the approach in practice. One of them was even enthusiastic about the idea, but the initiative would have to come from the customer.

Two suppliers could readily fill out their part of the use cases (the proposed solution), and we had no problem understanding their proposals. They had described advanced solutions to the various problems, and they explained that this was a very good place to write about the advantages of *their* solution.

The first supplier gave the task of filling their part of the use cases to another staff member. Apparently this staff member didn't get the proper instructions, so he misunderstood the idea and wanted to correct the description in the left-hand column. As an example, he complained that the customer in step 4b (Fig. 3) had a problem with access to available staff in other *departments*. Why didn't he ask for access to available staff in the entire *county*? (Their product offered that). Actually, he could have written that as a proposed solution in the right hand column, thereby showing that they had a solution that exceeded what the customer asked for.

In: Proceedings of REFSQ'99, Heidelberg, June 1999-06-24

Conclusion

We have tried to apply the use case idea in a COTS tender. The aim was to replace most of the present functional requirements with use cases. We used high-level use cases and modified them in several ways so that they could describe problems in the present way of doing things, describe a possible solution, and leave space for the supplier to specify his solution. Once the principle was developed, the researchers could specify the actual use cases within a day.

Evaluations and reviews showed that suppliers as well as customer got a better understanding of the requirements and found them less supplier dependent. It was easy to trace project goals to requirements, thus improving the chance that the system met the expectations and not just the stated requirements. Several important requirements were revealed through the use cases, while the existing spec didn't catch them. However, some existing functional requirements were difficult to cover with use cases, because it was hard to identify tasks where the requirements were needed.

Suppliers understood the approach within three minutes. All suppliers were willing to try the approach, and one even suggested that we persuaded customers to use the approach in future tender processes. In order to do that, customers need encouragement and training. Suppliers suspected that the use case approach would be more time consuming than the traditional approach. Whether they are right has to be tested in real life.

References

Checkland & Scholes: Soft Systems Methodology in Action. Wiley, 1990.

Cockburn, A.: Structuring use cases with goals. Journal of Object-Oriented Programming, Sep-Oct 1997 & Nov-Dec 1997. Also in: http://members.aol.com/-acockburn/papers/usecases.htm

De Marco, T.: Structured Analysis and Systems Specification. Prentice-Hall, 1979.

Graham, I.: Requirements Engineering and Rapid Development. Addison Wesley, 1998.

Jacobson, I. et al.: Object-oriented software engineering - a use case driven approach. Addison Wesley, 1994.

Kovitz, B.L.: Practical Software Requirements. A Manual of Content & Style. Manning Publications, 1999.

Larman, C.: Applying UML and Patterns. Prentice-Hall, 1998.

Lauesen, S., Harning, M.B., and Grønning, S.: Screen Design for Task Efficiency and System Understanding. In: S. Howard and Y.K. Leung (eds.): OZCHI 1994 Proceedings, pp 271-276.

Lauesen, S.: Usability requirements in a tender process. In Paul Calder & Bruce Thomas (eds.): OZCHI'98 Conference Proceedings, IEEE Computer Society, 1998, pp. 114-121.

Meyer, B.: Object-oriented Software Construction. Prentice-Hall, 1997.

Schneider, G. & Winters, J.P.: Applying Use Cases. Addison Wesley, 1998.

Sommerville, I. & Sawyer, P.: Requirements Engineering, a good practice guide. Wiley, 1997.

Yourdon, E.: Modern Structured Analysis. Prentice-Hall, 1989.