LESSONS LEARNED FROM ASSESSING A SUCCESS

Søren Lauesen, Copenhagen Business School, Denmark
Jens-Peder Vium, Innovation and Quality Management, Denmark

Summary

This paper is a case study of a business application development. The system was developed after a tender process and based on a standard application. Management considered the system almost perfect: It was developed on schedule and within budget, and it improved business. In order to learn from the success, we conducted an assessment and identified an unusual requirement approach which in a systematic way specified functionality without designing the user interface.

We also identified three system problems that management believed were due to user resistance: (1) An important system goal was not supported by the system, although the goal was written in the requirements. (2) There was a specific usability problem in part of the system. (3) There were performance problems for certain large data sets. The problems could be traced to deficiencies in the requirement specification, combined with insufficient quality assurance during development.

In the paper we show the strong sides of the requirement approach, and we suggest how it could be further improved to prevent the three problems in a systematic and cost-effective fashion. A central part of the suggestion is a systematic way to trace system goals to requirements.

Søren Lauesen, Copenhagen Business School, Howitzvej 60,
DK-2000 Frederiksberg, Denmark. Tel: +45 38 15 24 27,
E-mail: slauesen@cbs.dk

Jens-Peder Vium, I & Q Management, Skindbjergvej 5,
DK-9520 Skørping. Denmark. Tel: +45 98 39 18 33,
E-mail: iqm@pip.dknet.dk

1. Background

A common type of development projects is business applications for small and medium sized companies. Such systems are often based on standard systems, modified and enhanced for the company in question. The standard system is selected during a tender process, where the proposer will supply the standard system and develop the modifications. This paper is a case study of such a development project in FSV (Fredericia Skibsværft A/S), a medium sized Danish shipyard which specializes in complex ship maintenance, rebuilding, and repair. In this case, development of the modifications took about one man-year.

What should the requirement specification look like in such cases? A central issue is how to describe the functionality of the system. One extreme is to prescribe the user interface as a set of screen pictures and menus, but this is really a design issue, and it would exclude proposers with a standard system deviating from the design.

In the shipyard case another approach was taken: The required functionality was described as a set of *scenarios* covering the different work areas to be supported. Each scenario gave some background information to allow the developer to imagine the work context. The scenario was further broken down into a set of *task steps* to be provided as system functions. (Jacobsen [1994] uses the term "scenario" to denote a specific task instance, while we use it to denote a group of related tasks).

The requirement specification was supplemented by a data model and examples of screen pictures in order to improve understanding of the information demands. This is a reasonable level of specification that allows the customer to verify the specification, and allows the proposer to estimate the degree of deviation from his standard system.

FSV has 150 employees and up to 350 temporary workers from subcontractors. Orders are known only one to three months in advance. Fast delivery is crucial for this kind of orders, and much of the work is not agreed upon until the ship is docked and inspected by FSV staff and a representative for the shipping company. The situation is thus quite dynamic, and efficient data registration and quotation calculation are essential.

In 1991 FSV decided to replace most of their business application for several reasons:

(a) Part of the system platform had to be replaced. The old system consisted of two different hardware platforms with loosely coupled software systems. One system was proprietary and could not be maintained anymore, so it had to be replaced. The other system was Unix/Oracle-based and parts of it had to survive and become integrated into the new system.
(b) Text-based documents and data base information should be fully integrated.
(c) Data should be up-to-date in all applications.
(d) Systematic marketing should be supported.
(e) Experience data from earlier orders should be available for calculation of new quotations.
(f) Invoicing and post-calculation should be speeded up in order to finish the administrative procedures while the shipping representative performs final inspection of the ship just before launch. (The ship cannot be launched until the invoice is ready, and a delay can cost the shipping company about US$40,000 a day.)

In October 1991, FSV had made some design suggestions in the form of screen outlines and print reports. However, these were just examples and could not be used as requirements in a tender process. FSV contracted with a consultant (Jens-Peder Vium, one of the authors) to assist in a tender process. At that time, the goals (a) to (f) above were not clear at all, and the consultant spent much time analyzing the current situation and setting up critical success criteria. Cooperating closely with FSV, he defined the goals above and made a requirement specification. In April 1992 the specification was sent to five suppliers that had shown interest in making a proposal.

Four suppliers submitted a proposal, but only one of them conformed to the requirements. Only this supplier showed a genuine interest and understanding of the application domain.

After a period of contract negotiations and joint design work, the contract was signed in July 1992. The original requirement specification with a few corrections was part of the contract and specified what to deliver. Delivery was scheduled to take place in several stages, with the last delivery to take place in April 1993.

The actual development was based on the supplier's standard accounting system, which used Oracle and could share data with the surviving system parts. A developer from the supplier worked full time on system extensions and data conversion in close cooperation with FSV staff.

In April 1993 the system was in operation - on schedule and within budget. Compared to typical development projects, this is unusually successful. Vium [1994] gives a detailed account of the course of events.

2. Observed problems

In 1995 - two years after system deployment - one of the authors (Søren Lauesen) became interested in the case, and together with a graduate student (Susan Willumsen) he reviewed the requirement specification and assessed the actual system. The aim of the assessment was to analyse the relation between actual system use and original requirements in order to identify good approaches and possible residual problems.

During the assessment, we identified three system deficiencies:

(1) One of the system goals (e) was not supported: Experience data from earlier orders should be available for calculation of new orders, but post-calculation data were not collected in such a form that this was possible. Further, the calculation of quotations was still based on the surviving system parts, so even if experience data had been collected from post-calculations, it would not be readily available for new quotations.

(2) Invoicing was a time-critical task, but it was not well supported: An invoice after a ship repair is a complex document with several lines of explanatory text for each item on the invoice. However, the full text was not visible on the screen. The system could show the text for only a single item at a time (in an auxiliary window), which made editing and review difficult. In practice, users had to print out the text several times for editing.

(3) There were performance problems when editing long invoices: When an invoice was more than 20 pages, the time to scroll from one end of the invoice to the other was unacceptable to users, and editing became cumbersome. We found many cases where invoices were more than 100 pages and where the sequence of items had to follow customer specifications, rather than the accounting sequences. So breaking an invoice into several smaller invoices was not an acceptable solution. (As non-domain experts, we were surprised that invoices could be that long, but the developer knew about it from the period of contract negotiations.)

It is interesting that management had heard about these problems, but did not consider them caused by the system. Management believed that the problems were due to user resistance against EDP in general, and they were surprised when we pointed out that the cause actually was system deficiencies. As it often happens, users do not agree on the importance of the problems: Management consider only problem (1) a major deficiency, while invoicing staff consider problems (2) and (3) major deficiencies. We do not want to elaborate on these matters, but will consider all three problems worth investigating.

Fortunately, the consultant had an attitude of continuous improvement, and he found the assessment an opportunity to improve his techniques for requirement specification. Would an improved requirement specification have helped? Why had the problems not been detected during development and installation?

3. The requirement specification

Before we discuss whether an improved requirement specification would have helped, we will illustrate the style of the actual requirement specification. It consisted of 30 pages plus 60 pages appendix, as follows:

Background information (3 pages): About FSV's business and existing computer systems. Purpose and audience for the specification.

Definition of terms (8 pages): An explanation of information terms used in the shipyard, like *job, activity, dock list*. A definition of generic, user-oriented functions in the system, like *filing* and its variations, *skimming, data entry*.

Purpose of the system (1 page): The system goals, roughly corresponding to points (a) to (f) above. Overall requirements like *decentralized system use* and *terminals gradually to be replaced by PCs*.

Functional requirements (12 pages): A list of scenarios covering the different work areas to be supported, e.g. *marketing and sales, quotation calculation, job scheduling, invoicing*. For each scenario the necessary user-oriented system functions (task steps) are listed. Fig. 1 shows an example of a scenario description. The functional requirements comprise about 20 such scenarios.
Soft (non-functional) requirements (1 page): Required availability and response time (elaborated in section 5).

Other requirements (5 pages): Modes of operation (e.g. *normal operation, batch mode, restart mode*). System and user documentation. Project management and quality assurance. Installation. Acceptance test.

Appendices:
Existing, surviving system parts (14 pages): Data models. Selected screen prints.

New data model (14 pages): New, extended data model. Description of crucial algorithms for calculation etc.

Design outline (25 pages): Outlines of suggested new screens. Outlines of task steps to carry out some critical tasks. Screen prints for all screens in the old system. It is explicitly stated that this design is just an example and not a requirement.

Other (6 pages): Life cycle of an order. Relation between order, invoice and account numbers.

4. Strong Sides - Scenarios

In a tender process like FSV's, the requirement specification must be understandable to a customer with rather little EDP expertise. At the same time, it cannot specify functionality as a set of screen pictures and menus, since that would exclude proposers with a standard system

Invoicing

The reviewed cost transactions are transferred to the invoicing staff. Invoicing requires business knowledge and flair in order to ensure that the invoice becomes what it should end up with, rather than a basis for discount discussions - or what could be worse.

Invoicing comprises the following functions:

a) Printing of cost transactions with costs per job, job text, totals per cost type. See example in app. xxx.
b) Entering calculation factors like profit percent and prices per hour. These values apply only for the order in question. Default values are available from the customer record.
c) The invoice process: Jobs are grouped into invoice jobs. Each invoice job appears on the invoice with a total per cost type. Possibility for editing all information except the job number. Possibility for adding comments. It must be possible to temporarily close the function so that it can be resumed later, for instance the next day.
d) Printing of finished invoice. Printout of temporary invoice with possibility for specification to the individual job level. See example in app. xxx.

Invoicing must further have access to the following functions:

- From sales domain: Function d (queries on country/customer/ship/order), function g (queries on quotation and experience data), . .
- From job scheduling: . . .
- From . . .

Figure 1. A sample scenario description with an introductory background and lists of associated task steps and user-oriented functions.

deviating from the design. What is the standard approach for specifying functionality and usability? Davis [1990] is probably the standard reference in this area. He suggests prototypes of screen pictures and menus as the only viable solution (p. 337), but it would not work in this case.

The solution chosen in the shipyard case was to specify the functionality through a set of scenarios describing the work areas (Fig. 1). ?Scenario? is a term used with many meanings (Campbell, [1992]). Jacobsen [1994] uses the term to denote a specific task instance, e.g. a specific order to a specific customer, while we use it to denote a group of related tasks. An important point in our scenarios is to include an introductory background which gives the flavour of the work area. The scenario also gives a list of task steps and user-oriented functions to be available. In most cases, the task steps do not have a specific sequence, since sequences vary a lot depending on the actual situation.

The shipyard specification is sufficiently short for the customer to read it, and it uses terms entirely from the customers world. The design outline gives the customer a feel for what the system could look like, although the actual system may differ significantly. In principle, the customer should be able to verify that the list of scenarios and associated task steps is complete. However, as we shall see later, some crucial task steps were omitted in the shipyard case.

At the same time, the specification allows the supplier to estimate the degree of deviation from his standard system. The data model can easily be compared to the standard system, and the task steps compared to the standard functionality. Since the user interface is not specified as screen pictures, the supplier can reuse his standard system to a large extent. The scenario background and the design outline give the supplier improved confidence that he has understood the business problem correctly.

Finally, the functional requirements can be used during development and acceptance test to verify that everything is delivered.

In the shipyard case, the final screen pictures were quite close to the design outline. According to the customer, the major deviations from the design outline actually gave more useful pictures. Delivery took place in stages, and the customer checked the system with real-life data, using the scenarios and task steps as a checklist.

5. Weak Sides after a Brief Review

Our assessment - two years after system deployment - started with Lauesen making a one hour review of the requirement specification. He noticed the strong sides of the scenario approach for specifying functionality. He also noticed that availability was well specified, but other soft requirements were very vague. Response time, for instance, was only specified like this:

> Response times must not be that long that employees experience them as stressing. For data entry, no delay is acceptable at all. The response times must be realized in situations where 20 users work concurrently.

In principle, the absence of stress can be verified, but not until deployment of the system. We would prefer requirements that are more suitable during development, for instance response time with worst case data. Other aspects of usability like ease of learning and task efficiency were not specified. (Stress is one aspect of task efficiency, but a task can be unstressing and still take an unacceptable long time to perform.)

The contract was based on the original requirement specification with a few changes.

Interestingly, one change was that the line about *stressing response time* was cancelled. This seems reasonable, since the supplier cannot alone be responsible for user stress. However, no other specification of usability was added. Actually, the consultant would have liked to specify something, but found no help in current or best requirement practice.

Other soft requirements like data volumes, security, portability, and maintainability were not mentioned.

Lauesen also noticed that it was difficult to see which tasks were to be supported by means of the surviving system parts and which by means of the new system.

The scenario/task step approach is fine, but user tasks themselves are rarely mentioned. The job scheduling scenario, for instance, just specifies that the user must have access to a long list of functions. A frequent task in this scenario is to find the next job for a specific worker and print out the job description for him. To carry out the task, the user has to use several system functions. The total number of screen pictures and the total time spent in the task are critical for usability.

Tasks are also well suited for understanding the user's actual work with the system; for describing the system state before and after the task; and for making test cases for acceptance testing.

What Lauesen did not notice during the one hour review was that the system goals were not completely reflected in the specification. This turned out to be the root of the three deficiencies, as we learned after a site visit and further reviews.

6. Cause and Cure of the Deficiencies

We will now try to identify the cause of each of the deficiencies, and suggest techniques that could have prevented them or detected them before the contract was signed.

(1) Experience data not available for new quotations

The requirement specification stated the availability of experience data as a major system goal (goal e). Why was it not supported? It turns out that the quotation scenario mentioned the use of experience data, but without specifying what it was. Furthermore, an appendix to the requirement specified that the old system parts had to be used for quotation work. However, the old system did not support use of experience data.

No scenario mentioned system functions for collection of experience data. Probably, the intention was that all data collected was a kind of experience data. However, a closer look at the real demand showed that key ratios like *work hours per ton of iron* had to be collected, but no provision was made for that.

In summary: There was no traceability from system goals to system functionality. Davis [1990] mentions the need for traceability from demands to requirements (p. 193), but gives no specific technique for handling it.

In our experience, this is a common problem. Many requirement specifications suffer from such a lack of traceability between system goals and functionality. In the shipyard case, the contract referred to the (modified) functional requirements, and explicitly omitted the system goals which were still part of the specification. In other words: The supplier could not be kept responsible for the system goals.

It would have been quite easy to prevent this problem. A review of the requirement specification could have included a manual walk-through to trace goals to functionality. With goal (e), we would have asked: Which system functions would be used to satisfy this goal? And the result would have been: The quotation scenario mentions the use of experience data, but a function to show the experience data is not part of any delivery. No function specifies how to collect the data.

A more systematic way to trace the goals is to make a matrix that relates goals to requirements. Fig. 2 shows part of such a matrix for the shipyard case. To the left, we have the six system goals, and at the top we have the major parts of the requirement specification:
The scenarios (functional requirements) and the soft requirements (usability and response time). A dot in the matrix shows that a goal should be reflected in a part of the requirements, in the sense that the new system should provide more than the existing system.

If we look at the goal *experience data for quotation*, we see that it requires new functionality for quotation calculation, cost registration, and invoicing. It is now easy to inspect the corresponding requirements and detect that the new requirements are not mentioned correctly.

The first goal, *platform replacement*, shows the functions that had to be replaced because they ran on the proprietary, old system. Note that the quotation calculation did not have to be replaced for this reason. However, any function with a dot in its column had to be modified somehow, so quotation calculation had to be modified or replaced.

	Requirements									
System Goals	Marketing and sales	Quotation calculation	Job scheduling	Cost registration	Invoicing	...	Payroll	EDP operations	Usability	Response time
(a) Platform replacement				●			●	●		
(b) Text and database integrated	●			●	●					
(c) Data up-to-date			●	●	●					
(d) Marketing support	●	●								
(e) Experience data for quotation		●		●	●				●	
(f) Invoicing speeded up		●		●	●				●	●

Figur 2: The matrix for tracing system goals to requirements.

This means that the matrix would have addressed Lauesen's early observation that it was difficult to identify surviving and new system parts. As a side remark, we realize that the matrix looks like a QFD quality house, but we use it for tracing goals to requirements, rather than for prioritizing solutions. (Brown, [1991]; Hauser & Clausing, [1988])

(2) Usability problem when editing invoices

System goal (f) specifies that invoicing should be speeded up, but again this is not easily traced to system functions. In the old system, numeric data was entered during order execution, but descriptive texts were transferred on various paper forms and entered at invoice time by means of word processing. In the new system, the descriptive texts had to be entered during execution of the order, for instance as comments to the quotation or the cost registrations. In this way, most texts are immediately available for invoicing. The matrix reflects this, but review of the corresponding parts of the requirements shows that cost registration does not mention text entry. This requirement deficiency caused no problem, however, since the developer understood the situation and implemented the facility.

The observed system problem here has nothing to do with the functional requirements, but the soft requirements, in particular the usability requirements. Since fast invoicing is stated as a major goal, it would have been reasonable to spend some effort in specifying usability requirements for invoicing. Again, the matrix would probably have drawn attention to this issue.

One solution would have been to study the invoice task more closely during the requirement stage. Observations of users would probably have revealed that they do a lot of editing of invoice texts, correcting spelling errors, comparing texts from one item with another, etc. A requirement could then have been set up like this:

> Requirement A: During invoicing, the user must be able to see and edit the invoice like word processing, where the entire text is visible.

An alternative could have been to specify usability on a higher level as a performance requirement:

> Requirement B: The invoice staff must be able to compose and edit an invoice like appendix xxx in yyy minutes. (Appendix xxx would contain a typical invoice of average size.)

This requirement is much stronger than the first one since it comprises all user actions and system response times. In order to fulfil such a requirement, the supplier would have to make early, functional prototypes and test them against the usability requirement. Delaying the usability test until delivery time would be too risky for all parties involved.

Best practice in requirements engineering has little to provide for specifying usability (Hsia et al., [1993]). Our inspiration for the suggestions above comes from the human-computer interaction field and an example in the IEEE Guide (IEEE, [1984], section 6.3.1.5.1).

(3) Performance problems for long invoices

Scrolling long invoices has an unacceptable response time. This problem is closely related to the usability problem, but it also reflects that data volumes are not specified in the requirements. Although the supplier and manager knew that invoices could be a few hundred pages, they only tested the system with short invoices. An explicit requirement could have drawn attention to the problem and given the supplier the responsibility for resolving it. A requirement like this would have worked:

> The system must be able to handle invoices of up to 500 pages with a total of 10,000 items. The time to scroll an invoice or search for a specific item must not exceed 5 seconds.

An alternative would be to cover the requirement like usability requirement B above, for instance specifying a worst case invoice in an appendix. The matrix of fig. 2 might have helped here too, since the goal *invoicing speeded up* would probably be related to the response time requirement.

Balancing the requirements work against QA during development

If we were to follow all the guidelines for good requirements (e.g. Davis, [1990] or Birrell & Ould, [1985]), the specification work would be out of proportions compared with the system size. Further, the specification would be too long for careful customer review and acceptance testing. An alternative would be to use quality assurance during the development process to detect requirement deficiencies.

However, in contract development like the shipyard case, it is risky to detect requirement deficiencies late. The supplier may refuse to correct the problem or may ask for an unacceptable high price for the correction. In such a case, the customer may be unable to reject the system and terminate the contract (paying nothing), since the supplier fulfils the stated requirements.

As we have shown above, a rather modest effort could have improved the requirement specification significantly, thus reducing the risk:

1. Manually trace the system goals to requirements.
2. Specify usability goals for critical user tasks.
3. Specify worst case data volumes and response times.

We believe that these techniques could have been used without disturbing the good sides of the development approach. They would also have provided strong test cases for the acceptance test.

Why were the problems not detected during development?

In the shipyard case, it is surprising that the problems were not detected during development or initial deployment. The reason seems to be an adverse side of an otherwise successful development approach:

During development, the supplier and customer management worked closely together and became a team with a common goal: Deliver a useful system on time and within budget. This team spirit is probably the prime cause for on-time delivery (Keil & Carmel, [1995]; DeMarco & Lister, [1987]).

On the other hand, this made supplier and management blind to the system deficiencies, as it normally happens in development teams. The daily users now became an alien party with little say in the process. Management turned user complaints down, believing that "users were just reluctant to use EDP".

An independent quality assurance function would have helped here, for instance to track requirements during development, and to ensure that worst case data were used for testing.

7. Lessons Learned from the Case

In the shipyard case, we have learned some successful techniques and some weak points that caused significant problems. We have investigated the weak points and come up with cost-effective techniques that could have complemented the successful techniques and prevented the problems.

In summary, we suggest the following techniques for contract development of business applications that are based on modified standard applications:

1. Specify the system goals: What are the purposes and expected benefits of the system?
2. Specify the functional requirements as scenarios for the various application domains, giving the supplier a feel for the environment and work conditions. Specify critical tasks for each domain, and task steps (system functions) for the entire domain.
3. Specify usability requirements for the critical tasks, for instance as total task performance times including user actions and system response time.
4. Make a walk-through for each system goal to see that it is reflected by the functional requirements and usability requirements. Or use a matrix relating goals to requirements.
5. Specify worst case data volumes and response times for critical tasks. (The ideal is to specify it for all tasks, but that would be overkill).
6. Illustrate the functional requirements by means of a data model (e.g. an E/R-diagram) and outlines of possible screen pictures. These are examples to help understanding - not requirements.
7. Encourage close cooperation between supplier and customer, but ensure that independent quality assurance is in place for early discovery of deficiencies.

Acknowledgements

We would like to thank Orla Skjellerup from FSV for his patience in explaining and showing us the actual system and work environments, and Susan Willumsen for her cooperation and sharp observations in the case study.

References

Birrell, N.D. & Ould, M.A.: A practical handbook for software development. Cambridge University press, Cambridge, 1985.

Brown, P.G.: QFD: Echoing the voice of the customer. AT & T Technical Journal, March /April 1991, pp 18-32.

Campbell, R.L.: Will the real scenario please stand up? SIGCHI Bulletin, April 1992, pp 6-8.

Davis, A.M.: Software requirements - Analysis and specification. Prentice Hall, 1990.

DeMarco, T. & Lister, T.: Peopleware - Productive projects and teams. Dorset House Publishing, 1987.

Hauser, J. & Clausing, D.: The house of quality. Harvard Business review, May-June 1988.

Hsia, P., Davis, A. & Kung, D.: Status report: Requirements engineering. IEEE Software, November 1993, pp 75-79.

IEEE Guide to software requirements specifications. ANSI/IEEE Std. 830-1984.

Jacobsen, I.: Object-oriented software engineering - A use case driven approach. Addison-Wesley, 1994.

Keil, M. & Carmel, E.: Customer-developer links in software development. Communications of the ACM, Vol 38, No. 5, May 1995, pp 33-44.

Vium, J.-P.: Installing IT right the first time. European Quality, Vol. 1, No. 5, 1994, pp 34-37.