Accepted for presentation at REFSQ 2012. Will be published in Proceedings of REFSQ 12, Springer Verlag.

Why the Electronic Land Registry Failed

Soren Lauesen

IT University of Copenhagen, Denmark
slauesen@itu.dk

Abstract. [Context and motivation] In 2009 Denmark got a compulsory IT system for Land Registration of ownership. It soon created a national disaster because selling houses and getting mortgages might take months, rather than a couple of days. In this period, house owners had to pay a much higher interest rate. **[Question/problem]** The press claimed it was yet another IT failure, but actually the IT system worked as intended. What was the real cause? **[Principal ideas/results]** The visible problem was overloaded staff in the Registry Office, but behind this were optimistic estimates of human performance, lack of usability, insufficient user interface requirements, unrealistic SOA requirements, immature risk analysis, and other factors. **[Contribution]** This paper shows details of the requirements, what went wrong, and what could have been done, e.g. early design of the user interface and giving the supplier more influence on the architecture.

Keywords: information system failures; software failures; public software acquisition; organizational implementation; usability; user interface requirements; SOA architecture; risk analysis.

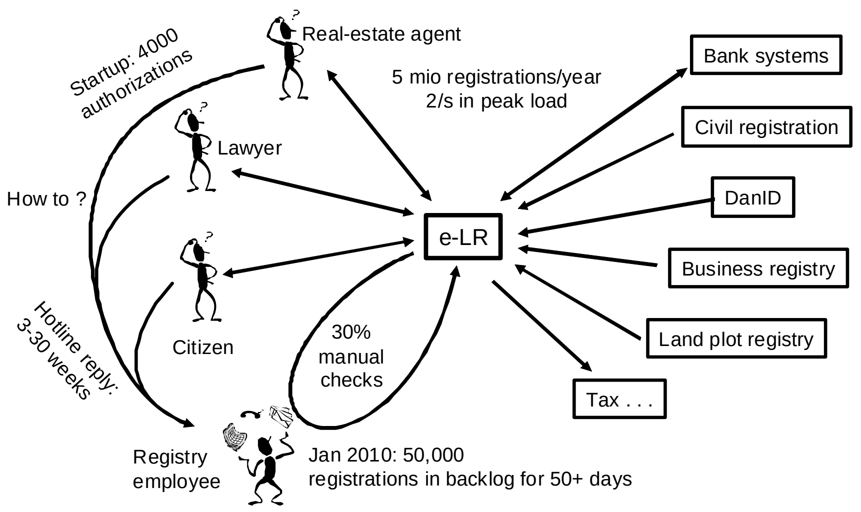
1 Background

In September 2009 Denmark got an Electronic Land Registry system (e-LR) that overnight became the only way to register ownership of land and mortgage deeds. All registrations had to be entered online or through system-to-system interfaces to financial IT systems. It was planned that 30% of the registrations should be selected for manual checking based on confidential criteria. All stakeholders had known about the new system for a long time and had been advised to prepare for the change.

Figure 1 shows the system and its context. Real-estate agents, lawyers and ordinary citizens were expected to use a web interface (the *external portal*), the registry staff used an internal user interface (the *internal portal*). The e-LR system integrated to several government systems, e.g. the national digital signature (DanID), the civil registration system, the business registry, one of the tax systems (for collecting the registration fee) and the land plot registry.

Based on historical data, it was estimated that 5 million registrations would be handled per year, corresponding to about 2 per second during peak load in daytime. (Denmark has around 5 million inhabitants). The customer (the Land Registry) had carefully estimated the number of employees needed to support this load, but the estimate turned out to be far too small.

Fig. 1. Electronic Land Registry – Big-Bang Sept 2009



Technically the system worked as intended, but the staff immediately became overloaded. Early 2010, more than 50,000 registrations were waiting for manual checking with a delay of 50 days. The lucky 70% who were not selected for manual checking, got their registration within a minute.

The unfortunate 30% lost money because they had to pay a much higher interest rate in these 50 days. Since selling and buying real-estate is financially stressful to an ordinary family, this loss could have serious consequences.

The situation could have been much worse. Due to the financial crisis in 2009, there were only half as many registrations as expected. If the expected number had occurred, the situation had become a true disaster.

This paper reports on how the overload could happen. There are many studies on systems that fail or vastly exceed the budget and deadline [1, 2, 3, 4, 5, 6, 7, 17], but the explanations are usually on a high level, such as poor project management, lack of senior management support, or lack of user involvement. Poor project management doesn't directly create a disaster. It is created through many specific flaws, such as ignoring certain risks or not watching what the system supplier is doing. These specific flaws are rarely reported, and probably vary a lot from project to project. However, one study goes into detail: Leveson & Turner's investigation of the Therac-25 accidents, where patients were seriously harmed when treated by an electron beam [11]. The cause was believed to be operator mistakes or faulty switches, but turned out to be an error in the central real-time software where concurrent activities communicated through shared memory locations without proper synchronization.

In this paper we start with the visible problem - the overloaded staff - and identify the main causes behind it. Some of the root causes are well known factors such as poor project management, but some other causes are not covered by literature. Unfortunately, in complex systems the network of causes is not a precise concept. Several factors contribute in each link of the chain [5, 11, 16]. All we can do is try to identify the major ones and their approximate relationships.

2 Project history

The e-LR system was developed and operated by a large software house that won the contract after a tender process. The customer was the Danish Courts. Denmark has around 30 courts and most of them operated a land registration office for properties in their own district. The plan was to locate the electronic land registration in only one of the courts, in that way saving around 220 staff.

The history was as follows.

- Early 2005. A consultancy company had made a business case that showed that the registration office could save around $20 M a year, and citizens and the financial sector around $80 M. (For simplicity, we have defined 1 USD = 5 DKK.)
- June 2006. The customer and his consultant had developed tender material (including the requirements specification) and made a request for proposals.
- December 2006. The customer selected one of the suppliers and they signed the contract. Expected deployment of the system was March 2008.
- 2007. It became obvious that the system couldn't be delivered on time. The new deployment time was changed several times.
- 2007-2008. The huge archives of existing registered documents (40 million pages) were scanned and filed, ready for the new system.
- Early 2009. The 220 staff were dismissed with the standard notice time of around 6 months. It was planned to be around three months after deployment, but the schedule slipped once more, and as a result these employees had left when the system was deployed.
- September 2009. The system was deployed as a big-bang. The registry office soon became the overloaded bottleneck.
- Late 2010. The registration office managed to get rid of the backlog, and from the end of 2010, 99% of the manual checks were handled within 15 days. The remaining cases were the very complex ones.

The development time had increased from the planned 18 months to 36 months. The software cost had increased from the budgeted $15 M to $21 M (+40%).

Today the system is a definitive success. As an example, a citizen can walk into his bank, and in less than an hour get a mortgage and $200,000 on his bank account.

3 Method

Late 2009, the Danish State Auditors (*Statsrevisorerne,* members of the parliament) asked the National Auditors (*Rigsrevisionen*) to audit the project. They contracted with Lauesen to help them with the IT aspects. The team agreed that an important aspect of the audit was to identify issues that other IT projects could learn from.

The team gathered information in several ways. We read the existing documents about the system. From an IT perspective, the most interesting ones were:

1. The requirements specification (406 pages with 413 requirements).
2. The supplier's proposal (600 pages).
3. The contract (32 pages with requirements and proposal as appendices).

4. Design specification, change specifications, etc. (more than 2000 pages).

We interviewed real-estate staff and lawyer staff to hear about the system and the problems it caused. They also showed us how the system worked and how they used it.

We conducted a focus group with senior representatives for the stakeholders: the financial sector, the land surveyors, the lawyers, the real-estate agents, the customer (the Land Registry) and the customer's consultant. We asked about good and bad points in the system, and the stakeholders' priorities for improvements.

We had expected that the ordinary staff member's opinion differed from the senior representative's opinion, and that stakeholders disagreed with each other. This turned out to be wrong. Everybody agreed on good and bad points, although they gave them somewhat different priorities.

We met with experts from the financial institutions to hear about their experiences with the system-to-system integration, and with senior representatives for the customer and his consultant.

Later we met with the supplier's senior staff and developers to discuss our findings and the relationship between supplier, customer, and the customer's consultant. This brought a quite different perspective to what had happened and why.

We wrote our findings as a preliminary report and submitted it to the customer for review, discussed disagreements, and published the final report [15].

Later, Lauesen interviewed and exchanged mails with the president of the Danish Courts and the supplier in order to get further insight into the overload and related issues.

4 What caused the overload?

Why did the registry become overloaded? We found several reasons, but the most important were these:

Cause 1: An unexpected number of requests for authorization. It was expected that citizens would sign the various registrations digitally, using DanID, but few did. Although DanID had been operational for several years, few citizens used it. It was far too complex to install and maintain. (In contrast, most citizens used the bank's shared digital signature without trouble.) The result was that lawyers and real-estate agents needed to digitally sign on behalf of their clients, but to do so required authorization from the Registry. The Registry was surprised to get 4000 authorization requests, and handling them caused much trouble.

Cause 2: An unexpected, huge number of requests to the Registry's hotline. The requests came from lawyers and real-estate agents who couldn't figure out how to use the system.

Cause 3: Registry staff was much less productive than expected. They were not comfortable with the user interface, although they had received training.

Cause 4: Mistakes in recent registrations. Since the old registry staff had been dismissed, registrations until the big-bang were done by temporary staff, who made many mistakes. After the big bang, many of these mistakes were revealed during the manual checks and caused further delays.

Cause 5: Big-bang without a pilot test. Could the causes above have been anticipated? In hindsight it looks possible, but systems of this kind are so complex that there always are surprises. One way to detect them is by running a pilot test, for instance deploying the system in only one of the 30 Danish courts. This would have revealed all the causes above, and it would have been far easier to deal with them.

Below we will discuss the secondary causes, e.g. why the customer (the Land Registry) didn't ensure proper usability, which would have reduced cause 2 and 3.

5 Usability and user interface requirements

Cause 2 and 3 above are consequences of low usability, so we will look at the usability and how it was handled in the project. By definition we have a usability problem if the system can support the user's tasks, but the user is unable to find out how or unable to perform the tasks efficiently [10, 12, 13].

There are some examples in literature where low usability seems to be the root cause of the system failure, e.g. the London Ambulance [1], the FAA Air Traffic Control System [5] and the Sydney health record system [16].

Here are four of the many usability problems real-estate agents and lawyers told us about in the e-LR system:

1. *How do you register a condominium deed? There were several options in the menu: Single-family housing, cooperative apartment, farm – but no condominiums.*

The professionals were stuck and called the e-LR hotline. It was busy, so they might wait for an hour and then try the next day at a different time. It might take three weeks to succeed. Once they got through, there was an immediate reply: Select single-family housing – it includes condominiums. Since hotline had got this question frequently, one might wonder why developers didn't change the menu. The reason was that the judge in charge of the entire project refused: the law was clear and the term single-family housing covered also condominiums.

Amazingly, the Land Registry was not aware that the essential waiting time was three weeks. To his staff, it looked as nobody waited for more than an hour.

2. *Is it free to make a trial registration? And how much does it test?*

The user interface offered *trial registration*, but there was no hint at what it did and whether there was a fee. Professionals knew that there was a fee (a tax) for registering a mortgage, but would they be charged already at the trial? They had also experienced that a registration was accepted in trial mode, but rejected when they tried to make it final. So what was checked? Again it would take a long time to get a reply from hotline.

3. *"Registration rejected". But why? Just try again?*

When a registration was rejected, there was no explanation of the cause. Professionals experimented wildly to find out why.

4. When are requests selected for manual checking?
The system told you that your request had been picked for manual check, but professionals had no idea why. When time approached Xmas, some professionals wrote "Merry Xmas" in the field "message to the registry staff" - just to show sympathy with the Registry staff. They didn't realize that the result was that the registration was picked for manual checking (otherwise the staff couldn't see the message). The consequence was a delay of two months.

Usability testing
How could these usability problems have been avoided? Usability specialists recommend that you make usability tests where potential users try to perform realistic tasks with the system [10, 12, 13, 14]. They are given the same help as they would have in real life. One or two usability specialists or developers observe what the user does and record the problems. To help them understand the problems, the users are asked to think aloud. Next the development team tries to remedy the serious problems, and then run the test again until the result is acceptable.

With this approach, all four usability problems above would have been easy to detect. The first two usability problems would also have been easy to repair even a few days before the big-bang. The last two problems are harder to deal with. They might require much programming. The last problem would even need some strategic rethinking, because some rules for picking a registration were measures against tax evasion, etc. So these rules had to be secret. But others could be open to help the users.

Usability experts also recommend that designers make an early prototype or mockup of the user interface. It is used for testing and improving the user interface. In this way, it would also have been possible to deal with the last two problems.

Usability requirements
What did the requirements say about usability? The main requirement was this:

> *Req. 153: The supplier must during development test the usability of the external portal. The bidder must describe how.*

This is actually a great usability requirement, compared to what most requirements say about usability. The supplier's reply to this requirement is also great:

> *Reply to req. 153: [We will test with] Rolf Molich's principles from his book . . .*

Molich is a Danish usability specialist and he too recommends thinking aloud with early prototypes [12, 14]. However, the supplier's reply to appendix 21 about quality assurance interprets Molich's approach in a different way:

> *Reply to app. 21, quality assurance: . . . this means that the test manager guides the test participants through the tasks, asks explorative questions, and helps as needed.*

This completely ruins the approach because the help available in real life is very different from this test approach. In real life nobody is available for guiding the user and helping as needed.

Apparently, none of these approaches were carried out in the project. Five months before the big bang, we find this change note among several others:

Change 32, 30-03-2009: Usability tests are replaced by a very close dialog between [a supplier expert and a land registry expert]

This means that a domain expert (a land registration judge) and the supplier's designer defined the user interface, but didn't do any usability testing. Usability experts know that a user interface designed in this way is only understandable to a domain expert. And this turned out to be the case also in this project. The lawyers and real-estate agents didn't understand.

During our interview with 10 key participants on the supplier's team, they admitted that they didn't know what usability testing was and hadn't done any. They had made some *user testing*, which seemed to be more like the procedure suggested in the reply about quality assurance.

The information we gathered from the customer (the land registry) showed that usability testing was considered a nice thing to do at the end - if time allowed, but it didn't. The attitude was that it was the professional user's own problem to learn about the system. They had not taken the opportunity to make courses, etc. The customer (and the supplier) used as an excuse that the system was not intended for the ordinary citizen. It was hard to make them realize that the problems we reported were experienced by professionals, not by the ordinary citizen.

Concerning usability, the Danish Tax authorities are strikingly different. Tax rules are very complex, yet the Tax authorities have very successful web sites for reporting your actual income and your expected income. These sites are not compulsory, but the Tax authorities measure their success by how many citizens use the sites. In principle, the e-LR could have been launched the same way.

User interface requirements

While usability requirements specify the *quality* of the user interface, user interface requirements specify the *functionality* of the user interface. This can be done in several ways, for instance listing the functions that should be available or describing situations where the user will use the system. Both approaches were used in the e-LR requirements. *Use cases* served as a list of functions and *user stories* as descriptions of situations. Use cases as well as user stories come in many versions, but the versions used in the e-LR were not effective.

Fig. 2 shows part of a user story from the requirements specification. The full user story is 5 pages and in the story the user has to click a lot of buttons. It is a vivid scenario where you as a reader can imagine the situation. It is obvious that the writer has imagined a very concrete user interface with screens, pictures, menus and buttons to click. There are a total of 7 user stories for the external portal and 11 for the internal.

Are these user stories requirements? This would mean that the final system should have screens and buttons as described here. This would give the supplier little freedom and would mean that the customer had taken responsibility for usability. Fortunately, the specification says that these user stories are not requirements.

However, the main idea of a long sequence of clicks, questions and screens to fill, is visible in the final system. For instance it takes 22 screens to specify ownership of a property.

Fig. 2. From the requirements: User stories

Notification of division of property
Hansel and Gretel got married in 1989, but now they will divorce. Throughout the marriage they have lived in Hansel's house and they have agreed that Gretel stays.

Hansel logs on to www.landregistry.dk. A welcome text appears. There is text and picture for land registry of *real estate*, *car mortgage* . . . He can see an icon for *Information Center* . . .

Hansel clicks on the text *real estate*. Then he is shown a login picture . . . Hansel has his digital signature stored on his PC . . . He is asked whether he wants to register or ask about real estate . . . and his e-mail address . . . and whether he will work with information on ownership, mortgages, easements or other.

He selects ownership and is asked whether it is
- Final deed
- Final deed on several properties
- Deed upon purchase price payment
- (And four other options, including division of property)

In total 5 pages for this user story. The spec says it isn't requirements.

Fig. 3 shows a use case that describes how the user can make a test registration. It is a typical use case with an elaborate template with goal, precondition, post condition and exceptions. In this example it is just a lengthy way of writing this:

Use case A.5: Test registration
1. The user chooses a filled out registration.
2. The system performs a test registration.
3. The user gets the result.

An even shorter version is a requirement in traditional IEEE-830 style:

Requirement A.5: The system must provide a function for test registration.

The main problem is that in the e-LR use cases we see too little of the context. Although use cases are supposed to explain the context, they rarely do in practice. Some of the user-story aspects are missing. When would the user do this? And what will he do afterwards? Notice that a system that just reports *Registration rejected* fully meets the requirement expressed by the use case.

The specification contained 23 similar use cases for the external portal, for instance

Fig. 3. From the requirements: Use cases

USE CASE			
Name: Test registration	Actor: External user		Ver: 1.0 ID: A.5
Goal:	It must be possible to perform a test registration of a deed that has been filled out in the portal, in order that the user gets a quick reply whether the deed can be registered.		
Precondition:	The user has filled out a registration and wants to check it against the Land Registry		
Step:	Actor:	System:	Proposer's solution:
1.	Select the registration	The user must select the registration to be tested.	
2.	Test register	The system performs a test registration of the selected item.	Context missing
Exceptions:			
Post condition:	The user gets the result of the test.		

fill out registration, attach file, sign digitally. The internal portal had 31 use cases. A note added that the use cases were not a full list of the user interface requirements.

Although these specifications go too far in the design direction (the user stories) or don't cover the context of use (the use cases), they are actually quite good compared to average requirement specifications. Most requirements deal poorly with the user interface, and the traditional techniques offer no help.

Task descriptions are an alternative that combines the best parts of user stories and use cases [9, 10]. Fig. 4 shows user interface requirements for registration of ownership, expressed as a task description. Based on the interviews with the professionals, the author wrote this task description in half an hour.

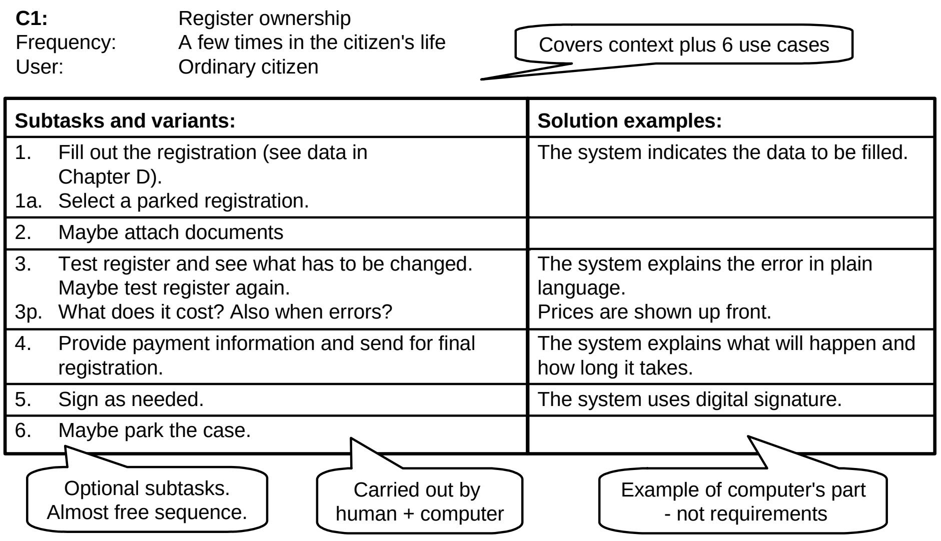
The left part of the task description lists what user and computer have to do together to register ownership. During a physical session on the web site, the user may do some of the steps, preferably in a free order. He should be able to park the case, for instance waiting for other persons to sign the registration. In this example, the system must enforce some preconditions between the steps. For instance it must not be possible to modify the registration without getting new sign offs. In order not to obscure the user's picture of the process, these preconditions need not be part of the task description.

The left part also mentions problems the user may have, for instance whether a test registration costs something.

The right part of the task description gives examples of what the system could do to support the user, for instance inform about the prices and what is wrong. The right-hand side is not requirements, but just examples. The true requirement is that the system must support this task (and maybe 20 other tasks).

Compared to the other approaches, this single task description covers the context, six use cases, and several usability problems. The description has not been carefully reviewed by domain experts, but the immediate reaction has been: *yes, we could have done it this way*.

Fig. 4. Task descriptions: The system must support C1 . . .



C1:	Register ownership
Frequency:	A few times in the citizen's life
User:	Ordinary citizen

Subtasks and variants:	**Solution examples:**
1. Fill out the registration (see data in Chapter D). 1a. Select a parked registration.	The system indicates the data to be filled.
2. Maybe attach documents	
3. Test register and see what has to be changed. Maybe test register again. 3p. What does it cost? Also when errors?	The system explains the error in plain language. Prices are shown up front.
4. Provide payment information and send for final registration.	The system explains what will happen and how long it takes.
5. Sign as needed.	The system uses digital signature.
6. Maybe park the case.	

Covers context plus 6 use cases
Optional subtasks. Almost free sequence.
Carried out by human + computer
Example of computer's part - not requirements

6 Architecture and SOA integration

One of the secondary causes of the staff overload was that development was late, so there was no time for a pilot test. This again had several causes, one of them being that there were time-consuming, unnecessary requirements in the architectural area. We will look at some of the causes here.

Availability and open target

The customer (and his consultant) had specified that the system had to be available 99.9% of the time. It is easy to ask for this, but customers don't think about the consequences. Experienced system operators can meet this requirement, but it is expensive. The system must run in several copies distributed geographically, maintenance and upgrades are complex, etc. If the customer had asked for 99.5%, it would be a routine matter.

In the e-LR case, the cost of operating the system with 99.5 availability is around $1 M per year. A 99.9% availability costs around $3 M per year. Is it worth it? In the old system, the availability was 25% because the Registry office was open 8 hours every weekday. Going for 99.9% in the future seems hard to justify.

The basic issue is that the customer may not be aware of the technical possibilities and their costs. This can be remedied by requirements with a more open target, such as this:

Req. 120: The customer expects availability around 99.8%.

The supplier could then offer two alternatives:

Req 120, alternative 1: 99.5% at 1M $ per year.

Req 120, alternative 2: 99.9% at 3M $ per year.

Such requirements and alternatives must be supported by rules in the contract, for instance as shown in [8]

Service-oriented architecture (SOA)

The customer (or rather his consultant) had suggested an advanced service-oriented architecture, and this was what the requirements asked for. We have summarized the requirements in this area as follows:

R1. The system must consist of modules connected with XML-services and a service broker. Each possible check of a registration must be a separate service.

R2. The system must connect to the external systems with XML-services. The data must always be retrieved from the external systems and not stored as a local copy.

A note added that all the external systems were stable and had well-defined XML interfaces.

These requirements sounded okay, but they caused many problems in practice. Here are some of them:

SOA eats computer power. Using an XML-interface requires 10-50 times more computer power (CPU time) than traditional approaches. With the high demand at peek load, this might become a problem. The supplier knew about this, but if he made reservations in his proposal, he ran a risk of being non-compliant. He ended up saying that he could make it as the customer asked for, but that he strongly suggested the traditional approach being used for the internal interfaces.

Not surprisingly, in the final system, the traditional approach is used internally.

Always getting data from the source degrades availability and response time. The reason is that if the external system is out of service, the e-LR system will essentially be out of service too. A similar argument holds for response time.

In this case the supplier made reservations in his proposal. The availability and response times in the external systems had to be "deducted" from the availability and response times of the e-LR system. The supplier also explained that he would construct the system so that it would be easy to change each external connection to a local copy with nightly synchronization.

Not surprisingly, the final system has a local copy of all the external data with nightly synchronization of changes. The only exception is the digital signatures in DanID. Here a high data actuality is justified, so that theft and other abuse can be stopped immediately.

In general, instead of asking for a specific architecture, the customer should ask for a specific data actuality, i.e. how old the data may be [8]. As an example, it is not a problem if a citizen's address is a few days old. This allows the supplier to come up with a suitable architecture.

The external systems were not stable. The customer's consultant's dream of stable external systems was just a dream. All of these systems (except the civil registration system) were under major revision. Furthermore, all of the systems had to accommodate changes made specifically for the e-LR system. These issues were very costly and time consuming to deal with for the supplier.

The supplier was lucky not to be judged non-compliant. Lauesen has seen some public acquisitions with unrealistic requirements such as 100% availability. All bidders except one made reservations. As a result the customer judged all of them non-compliant except the unrealistic one. Later in the project it turned out, of course, that the supplier couldn't meet the requirements, but the parties kept this confidential.

The ambitious SOA requirements were not really the customer's needs, but an idealistic concept enforced by the customer's consultant's IT architect. It took a long time to replace these ideals with something pragmatic.

7 Risk analysis

Several of the causes above could have been prevented with proper risk management. During the project the parties made regular risk analyses, but they seemed to be used mainly for arguing that the risk wasn't important. Most of the bad things that actually happened had been identified as a risk, but no action was taken. As an example, we find these risks early 2007 (abbreviated):

ID	Risk	Level: 5 highest	Consequence	Status/comment
1	SOA is immature	1		Tax uses SOA
2	Has the customer low IT experience?	1		Has much experience
3	Supplier staff leaves	3	Less time for test	Tight project management
4	Interfaces to many systems	3		The systems are stable

Comments:
Risk 1: The Tax department actually used SOA, but the large projects were not successful or not yet completed.
Risk 2: The customer (the Danish Courts) had experience with IT systems for internal use, but had not made a system for public use. With internal systems, they could easily support the users, but a system for large-scale public use was very different.
Risk 3: The supplier had planned to use a team with strong expertise in this kind of projects. However, the entire team was bought by Google. This stalled the project for a year, but the customer didn't notice. He just expected that the system would be delivered according to the contract. When the customer found out, he asked his consultant to manage the project. Together they succeeded making the supplier give the project a high priority.
Risk 4: As explained above, the systems were not stable.

Five days before the big bang, this risk analysis was made:

ID	Risk	Level	Consequence	Status/comment
5	Low usability shows up at deployment	[none stated]	Lack of usability	The case is closed. Probability reduced.
6	Lack of staff at customer site	4	Long delays	The customer assesses the situation.

Comments:
Risk 5: The status "the case is closed" refers to the agreement four months earlier about usability being replaced with a close dialog between customer and supplier. It is scaring that the consequence of low usability wasn't understood: high load on hotline and low productivity in the Registry office, causing further delay.
Risk 6: This is a clear statement that the risk is high, but the supplier will not take responsibility for the consequences. Earlier the supplier had recommended a pilot test and on-line help, but the customer claimed it was impossible.
It should be obvious that the risk analysis was not used correctly. There were no safeguards and nobody took action for the high risks.

8 Discussion and conclusion

Above we have identified many causes on various levels. Fig. 5 gives a graphical overview of them. We can see the network of causes and effects that resulted in low usability. We can also see the causes that made it impossible to run a pilot test. Notice that some causes have effect on several other causes, and some are caused by several lower-level causes in combination.

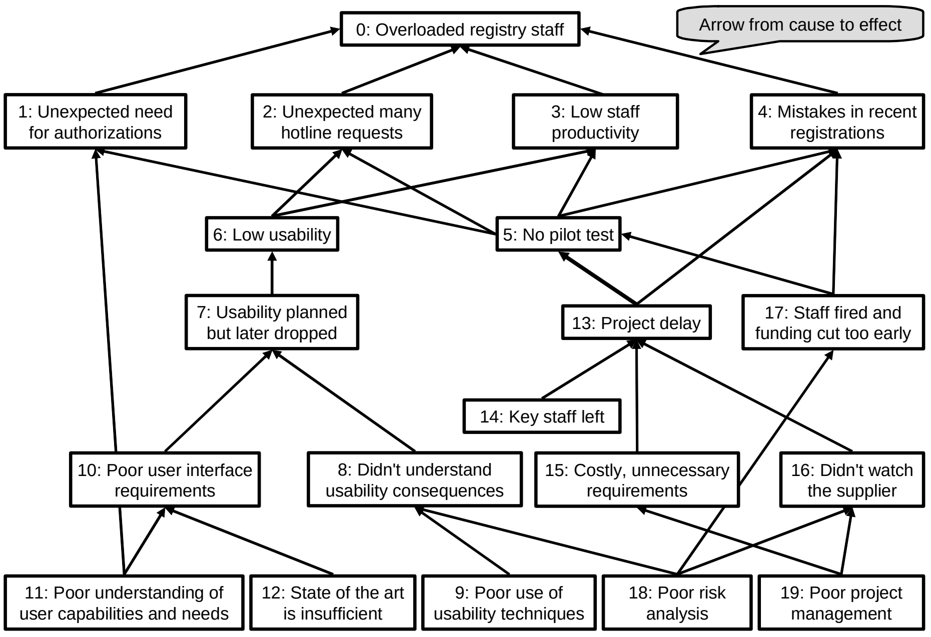
Cause 17 (staff fired and funding cut too early) has not been mentioned above. When a project like e-LR is funded by the government, the expected benefits are part of the project plan. Once the decision is made, the government cuts the funding according to the project plan. When the project becomes delayed, it is extremely hard to get the funding back. Further, if the customer waits too long to dismiss the redundant staff, he has to pay several months salary for no good. In the e-LR case, staff was dismissed so early and the project delayed so much that there was no time for pilot testing.

The figure also shows some broad root causes that were clearly in play here: Poor understanding of user capabilities and needs, poor use of established usability techniques, poor risk analysis, and poor project management. More surprisingly, a major root cause was that state-of-the-art in user interface requirements is insufficient.

Conclusion

The table below compares the findings in the e-LR project with the root causes reported by others. The list of root causes is compiled from Glass [5, 1998], Ewusi-Mensah [4, 2003] and Charette [2, 2005]. If we ignore the somewhat different word-

Fig. 5. Causes and effects


0: Overloaded registry staff
Arrow from cause to effect
1: Unexpected need for authorizations
2: Unexpected many hotline requests
3: Low staff productivity
4: Mistakes in recent registrations
6: Low usability
5: No pilot test
7: Usability planned but later dropped
13: Project delay
17: Staff fired and funding cut too early
14: Key staff left
10: Poor user interface requirements
8: Didn't understand usability consequences
15: Costly, unnecessary requirements
16: Didn't watch the supplier
11: Poor understanding of user capabilities and needs
12: State of the art is insufficient
9: Poor use of usability techniques
18: Poor risk analysis
19: Poor project management

ings, there is a large overlap between these lists. From the comparison, we can conclude this:

Human-performance causes are not covered by the literature: Most of the causes that directly related to the long delays are not covered by root causes from literature (causes 0, 2, 3, 4, 6). Although it seems obvious in hindsight, none of the authors suggest that you should estimate and test user performance, and that usability is a key factor for human performance.

Practices beyond state-of-the-art are needed: *Poor development practice (D)* and *poor requirements (B)* do in principle cover the issues about low usability and poor user interface requirements, but unfortunately the relevant techniques are not widely known. They are beyond state-of-the-art in software engineering.

Customers state ambitious requirements without caring about the cost: Although good requirements practice would guard against this, it doesn't happen in public acquisitions. It is all too easy for the customer to state very ambitious requirements, and in the tender process it is dangerous for a supplier to reply with an adequate, but less ambitious proposal. He might be deemed non-conforming.

The remaining causes are covered in principle, meaning that if project management, risk analysis, etc. had been carried out perfectly according to state-of-the-art, these causes should not occur.

Root causes from literature	**Causes found in e-LR**
A. Too ambitious project	15 (costly unnecessary requirements)
B. Poor requirements	10 (poor user interface requirements), 15 (costly unnecessary requirements)
C. Technology new to the organization	
D. Poor development practices	8 (didn't understand usability consequences), 9 (didn't master usability techniques), 12 (state-of-the-art is insufficient)
E. Inability to handle the project's complexity	
F. Poor performance by hardware/software suppliers	14 (key staff left)
G. Poor system performance	
H. Poor project management	5 (no pilot test), 16 (didn't watch the supplier)
I. Bad planning and estimating	13 (project delay)
J. Poor status reporting	
K. Poor risk management	7 (usability planned but later dropped)
L. Poor communication between customers, developers and users	1 (unexpected need for authorizations), 11 (poor understanding of user capabilities and needs)
M. Insufficient senior management involvement	
N. Stakeholder politics	17 (staff fired and funding cut too early)
O. Commercial pressures	
P. Causes not covered	0, 2, 3, 4, 6 (human performance)

Implications for requirements research and practice

There is a large gap between best practice in requirements and best practice in usability. As an example, early mockups of the user interface combined with usability tests of the mockups are not considered a crucial technique in requirements. It should be a standard approach.

Further, current practice is inadequate for specifying requirements to the user interface. Neither traditional shall-requirements, nor user stories or use cases are adequate. Task descriptions [9] cover much better and should be widely used.

Finally, there is a need to include user performance and organizational implementation in requirements. Little has been done in this area and literature is weak.

References

1. Beynon-Davies, Paul: Human error and information systems failure: the case of the London ambulance service computer-aided despatch system project. Interacting with Computers. Volume 11, Issue 6, June 1999, Pages 699-720.
2. Charette, Robert N.: Why software fails. IEEE Spectrum, Sept 2005. (Lists 31 failed US projects from 1992 to 2005.
3. Emam, Khaled El & Koru, A. Günes: A replicated survey of IT software project failures. IEEE Software, Sept/Oct 2008.
4. Ewusi-Mensah, Kweku: Software development failures. The MIT press, 2003.
5. Glass, Robert L.: Software runaways. Prentice Hall, 1998.
6. Jones, Capers: Patterns of large software systems: failure and success. IEEE Computer, March 1995.
7. Keil, Mark; Rai, Arun; Mann, Joan Elley Cheney; Zhang, Peter: Why software projects escalate: The importance of project management constructs. IEEE Transactions on Engineering Management, Vol. 50, No. 3, August 2003.
8. 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.
9. Lauesen, S. & Kuhail, M. (2011): Task descriptions versus use cases. Requirements Engineering Journal, DOI 10.1007/s00766-011-0140-1.
10. Lauesen, S.: User Interface Design - A Software Engineering Perspective. Addison-Wesley, 2005.
11. Leveson, Nancy G. & Turner, Clark S.: An investigation of the Therac-25 accidents. IEEE Computer, July 1993.
12. Molich, M.: Usable Web Design. Nyt Teknisk Forlag 2007, Denmark.
13. Preece, J., Rogers, Y. & Sharp, H.: Interaction Design – Beyond Human–Computer Interaction, John Wiley & Sons, New York, 2002.
14. Redish, J., Molich, R., Bias, R. G., Dumas, J., Bailey, R., Spool, J. M.: Usability in Practice: Formative Usability Evaluations — Evolution and Revolution. CHI 2002, April 20-25, 2002, Minneapolis, USA.
15. Rigsrevisonen: Beretning til Statsrevisorerne om det digitale tinglysningsprojekt (in Danish). August 2010. http://www.rigsrevisionen.dk/media(1610,1030)/14-2009.pdf
16. Southon, Gray; Sauer, Chris; Dampney, Kit: Lessons from a Failed Information Systems Initiative: Issues for complex organisations. APAMI/HIC'97 Sydney, August 1997.
17. Wallace, Linda; Keil, Mark: Software project risks and their effect on outcomes. Communications of the ACM, April 2004.