DamageCaseStories12.docx
Søren Lauesen 24-06-2024

Damages and damage causes in large government IT projects
Version 12

Soren Lauesen
slauesen@itu.dk

1. Land Registry 6
2. Travel card, RK 16
3. Police case management, PolSag 22
4. Debt collection, EFI 28
5. Health record system, EPIC 36
6. Cures for each cause 44
7. Cures 52
8. References 59

This report is intended for discussion and idea generation in a large audience. There may be errors in various places. Please send me a mail if you encounter errors or have ideas for preventing the damages.

The report consists of this document and a spreadsheet that gives overview. The latest versions are available at **http://www.itu.dk/people/slauesen/SorenDamages.html**
Earlier versions are available from the author.

Large IT projects are damaged in many ways, for instance large cost or schedule overruns, unsatisfied users, or disappointing business results. In spite of the damage, the new IT system is often deployed, but we would consider it more successful if it had avoided the damage. In some cases the project is closed because of the damages.

There are many reports on failed projects and suggestions on why this happens. Some causes are on a high level, for instance *bad project management* or *bad cost estimation*. Causes on such a high level don't really help us preventing the damage in the future. We might reason that since we have bad project management, we should educate better project managers. But such educations or certifications exist already.

Apparently they don't help. What do we have to add to these educations to make the participants successful?

Bad project management in itself doesn't kill a project. It is killed for technical or organizational reasons such as trusting new technologies too much, developing the user screens too late, or not noticing that the business results are about to disappear. Bad project management is when the project manager isn't aware of these factors and doesn't deal with them when they arise.

This report looks at 5 large, public Danish IT projects. For each project it summarizes the observed damages and the present state of the project. Next it identifies the observed causes of the damage. A troubled project has one to five observed damages and more than ten observed causes, each of which contributes to one or more of the damages. A project may have additional damages and causes that we haven't observed.

In this report, monetary amounts are specified in Danish kroner (DKK). Seven DKK are roughly one US$ or one Euro. Denmark has almost 6 million inhabitants.

The report identifies 37 damage causes, mostly observed in more than one project. It shows which project suffered from which causes and how the causes materialized in practice. Some of the causes correspond to well known recommendations. As an example, 13 of the causes were mentioned in a Danish report: Bonnerup/Teknologi-rådet (2001) [1]: Experiences from government IT projects - how to do it better.

For each damage cause, the report gives suggestions for what could have been done (the *cures*), for instance writing requirements in a different way, running pilot tests, monitoring the business aspects, etc. There is a list of 22 cures with a short description.

Sometimes a project encounters a damage cause that harmed other projects, but it caused no damage in this project. As an example, a project may have bad requirements, but the chosen supplier had a proven solution to the customer's problems - also to the problems the customer didn't mention in the requirements. In these cases we record the damage cause and explain why it had no consequences in this case.

The author has knowledge of the projects from many sources: published reports; his own experience in industry; work as a consultant for the National Auditors; reading piles of project documents; discussions with project participants; a broad professional network; supervision of students who come from a troubled organization and write a thesis about it; whistle blowers who tell facts that the National Auditors missed.

In its present stage, the findings and suggestions have been validated by managers and/or project staff from the specific project. This has taken place at meetings, seminars with many participants and in writing. After the first release of the report, I got several new facts and included them in the next version. One great surprise was that one of the projects (EFI) had been claimed closed by the press, but actually operated, although in a restricted form.

The purpose of this report is not to blame somebody. It is all too easy to look back and say *why didn't they notice? - they should have . . .* But in the real situation, people did their best, yet it ended with small or large damages. The purpose of this report is to learn from past damages in order to avoid them in the future.

The report consists of:

1. This paper, which gives the story behind each project and how the damage causes happened. It also lists all the 37 causes and the 22 suggested cures.
2. An Excel file with three spreadsheets: *Type-cure* with a row for each of the 37 damage causes. A column for each project; who failed; damage caused; cure type and suggested cures (if known). *Cures* with a row for each damage cause; a column for each potential cure and indications of which causes it would cure (at least partly). *VisibleDamages1, 2 . . .* with a row for each damage cause. Each project has several columns, one for each visible damage: The column indicates which cause contributed to this damage. The spreadsheets will print nicely on A3 or A4 portrait sheets.

Damages

We have classified the observed damages in this way:

Time: The project was significantly delayed. Sometimes this means added internal costs or lost profit.

Cost: The amount paid to the supplier and/or internal costs grew significantly.

Business results: The business case became worse than planned. This can be less profit. It can also be other business factors, such as increased waiting time for patients, wrong medicine orders, or low stakeholder satisfaction.

User satisfaction: The usability of the system became bad.

Supplier loss: The supplier lost money due to project issues. Although the customer formally doesn't care about this, good practice should be win-win for customer and supplier. Below we have two projects where the supplier decided to pay significant amounts for something he hadn't expected to pay for (Travel Card and Police Case management).

Feasibility doubts: There was significant doubt whether the project was feasible, i.e. whether it could be realized in the planned way. This happened in two of the projects (Police Case management and Debt collection) and was one of the reasons for closing them fully or partly.

Cure types

In the spreadsheets, we distinguish between these cure types:

Familiar: The cure is widely known in development communities. However, it may be ignored and in this way cause damages.

Unfamiliar: The cure is not widely known.

Misinformed: The project used a “cure”, which actually made things worse.

Change log

Version 2: 05-05-2017. First published version.

Version 3: 21-05-2017 and minor changes 27-05-2017.

a. PolSag: Cause A2 The 1:1 requirement and the use of use cases are explained.
b. PolSag: Cause G9 deleted and G10 modified.

Version 4: 30-06-2017, 13-11-2017.

a. EFI: Several corrections to reflect that EFI wasn't closed as the public believed, but continued with partly manual debt collection. Also explanations of how debt was handled after close down, and of the system to replace EFI. Cause A5 explains more about system integration.
b. Minor changes to align the text with the spreadsheet.
c. EPIC: Trial operation described in introduction. Cause F3 now explains why the low human performance wasn't detected early on.
d. The old cause B1 (believes law blocks sound approaches) is moved to G11, since it also occurs when a project is closed down.
e. Renumbering B2-B5 to B1-B4, and C5-C6 to C4-C5.

Version 5: 20-07-2017.

a. Land registry: The references are moved to the end of the Land Registry chapter and are now not only references to papers but also a description of what we did during the investigation. The plan is to do the same for all the projects.
b. Some cause names have changed a bit.
c. EFI and EPIC: Some statements were wrong (e.g. 1000 employees dismissed). Now changed.
d. Many details in the spreadsheets have been changed.

Version 6: 23-07-2017.

a. Cures were described under damages, which made it hard to find cures that worked for many damages. Now damages and cures are split into two chapters. Damages are numbered as before and they refer to the appropriate cures. Cures are numbered in a way similar to damages.

Version 7: 02-10-2017

a. This version is no longer a draft. The organizations involved have had the opportunity to comment on it, and several did. New versions may be released.
b. Various editorial changes in many places.
c. Cure CG1 has got one more advice: Re-planning can also involve re-scoping.

Version 8: 21-12-2017

a. This version has many updates about the health record system, EPIC.

Version 9: 14-04-1018

a. This version has a new cause definition for B4: Wrong selection criteria (earlier: Forgets costly items).
b. New facts about EPIC: It used wrong selection criteria. Although much needed, it wasn’t rescheduled because of heavy pressure from management as well as supplier.

Version 10: 23-04-2019 (DamageCaseStories10.pdf, by mistake it had the heading "Version 9")

a. The definition of familiar and unfamiliar cures has changed.
b. The description of cure CA1 (problem-oriented requirements SL-07) is new and includes a short comparison with user stories.
c. Reference chapter added.
d. Minor editorial improvements.

Version 11: 25-01-2020

There are now 37 causes of damage: two new and two combined to one. Two cures are added:

a. References for the Land Registry had disappeared. Now moved back.
b. Cause B2 and B4 have been combined to a new B2: Wrong selection criteria.
c. Cause G5 has been renamed to: The financial incentive disappears.
d. Cause G12, insufficient staffing, has been added. Observed twice.
e. Cause G13, doesn't find the root cause, has been added. Observed twice.
f. Two new cures have been added: CG7 (Ask expert developers) and CG8 (Problem-oriented contract).
g. In 2011 the Land Registry was charged in High Court for causing losses to citizens. The case was escalated to the Supreme Court in 2019. Details are added in the *method for data collection* section at the end of the Land Registry chapter. Explanation added of why 30% of the registrations were treated manually.
h. The state of the Health record system EPIC, hasn't changed much since its deployment in 2017 and no benefit has been reported. Details are added in the introduction to the EPIC chapter.

Version 12: 24-06-2024

a. P14. Ruling in the supreme court about low usability is now available.
b. Various misprints.

1. Land Registry

Denmark has around 24 court districts, each of which had a paper-based land registry. The idea behind the electronic land registry was to have only one land registry, which could handle most of the registrations on-line, in that way saving around 220 staff in total. In September 2009 the new system was deployed and overnight it became the only way to register ownership of land and mortgage deeds. All registrations had to be recorded online or through system-to-system interfaces to financial IT systems. It was planned that 30% of the registrations would be selected for manual checking, either because they were complex or as a check of randomly picked registrations.

Observed damages

a. **Time**: Development time was estimated to 1.2 years. Became 2.7 years. Due to the delay, the savings became around 120 M DKK less than planned. This is a major damage.
b. **Cost**: SW cost was estimated to 76 M DKK. Became 85 M DKK. Internal costs including scanning old paper documents was estimated to 190 M DKK. Became 233 M. We consider this minor damages.
c. **Business results**: The first 7 months after deployment, registry took 50 days for 30% of the registrations. The rules said that registration had to be within 10 days. Caused financial problems for a large number of citizens.
d. **User satisfaction**: The system was very hard to use and getting through to hotline took weeks in the first 7 months.

State today

Today the system is a success, not for the Land Registry only, but also for the financial sector. The planned benefit was met: 220 staff were dismissed. It is estimated that society further saves 200+ M DKK per year in handling costs. At the time of writing (2017), the system is still cumbersome and hard to use for ordinary citizens.

References

1. Lauesen (2012): Why the electronic land registry failed, 15 pages. In Proceedings of REFSQ 12, Springer Verlag. Also available at: http://www.itu.dk/people/slauesen/
2. Rigsrevisionen (National Auditors), August 2010: Beretning til Statsrevisorerne om det digitale tinglysningsprojekt (Report to the State Auditors on the Electronic Land Registry), 74 pages (in Danish).

Damage causes

Below we explain the observed causes of the damages in the Land Registry project.

Analysis

Cause A1. Doesn't identify user needs and win-win

In the old way of registering ownership, "users" sent paper documents to the registry and got papers back after some days. Now users had to do it on-line. Users were lawyers, real-estate agents, financial institutions, but also ordinary citizens. In order to serve them well, it is important to know their context of use, for instance how you get

two parties to sign, how you present the ownership at a board meeting, how you relate the registration to a loan application, how you review and correct errors, etc. Apparently there was little effort to investigate this. The Land Registry looked at it as a matter of filling in a form, signing it, and sending it with attachments. This was the "interface" they had in the old days.

In the requirements, ordinary citizens were also considered users, but early in the project is was decided to ignore them - it was too hard to deal with them. The project owners claimed that it had never been a goal to consider them. Requirements as well as other early documents say otherwise.

The financial sector planned for integration with their own systems. This too was not investigated, but 12 person months were set off for it. It turned out that 40 months were needed.

The result was that user needs were not addressed and it was unclear what the system should do from a user point of view.

Effects on damages: Business results, User satisfaction.

The registry project also included registration of car loans based on a new web-based car registration system. This was much simpler and the plan was to implement it first. But as the land registry was delayed and the new car registration system was in trouble, it was decided to implement car loans later. When "later" arrived, implementation of the car-loan system was painless. Asked why, the project owners explained that they had learned the lesson from the land registry: We started by finding out what the users actually needed. Then the rest was easy.

Cause A2. Requirements don’t cover customer needs

Compared to other requirements in large, public IT projects, the land registry requirements were remarkably short: 406 pages with 413 requirements. But like most projects, the requirements failed to address the basic question: who is going to use this system? when? and for what?

The Land Registry requirements try to deal with it in two ways: user stories and use cases. In those days, user stories were quite long, around 5 pages for a single user story. They described the user dialog in detail, what the user would see on the screen, what he clicked, etc.

There were 7 user stories for the public user interface and 11 for the internal, employee user interface. The requirements said that the user stories were not requirements, but for inspiration only. This "inspiration" is probably one reason that the final public user interface for recording ownership of a house takes the user through 22 screens.

The use cases were opposite. Each of them described a simple functionality the system should have. If a user wanted to register ownership, he would have to carry out many use cases: *login, fill in registration, attach file, sign digitally,* etc. Use cases don't show the context of use, and if you try to implement them literally, you will get a very inconvenient user interface. In total there were 24 use cases for the public part

and 31 for the internal. The requirements said that this was not a full list of the use cases to be supported.

Because there were no useful requirements to the user interface, it was hard to develop a user interface (user screens). Only in the last few months was this done - with disastrous results.

There were many other problems with the requirements that caused cost and development time to grow.

Effects on damages: Time, Cost, Business results, User satisfaction.

Cause A4. Makes heavy demands and believes it is for free

The customer (or rather his consultant) had suggested an advanced service-oriented architecture (SOA), and this was what the requirements asked for.

The system had to consist of modules connected with XML-services and a service broker. Each possible check of a registration had to be a separate service. The system also had to connect to around eight external systems with XML-services. Data had to be retrieved from the external source always, 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 costly and time consuming problems. 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 (2 registrations per second), this would become a problem. The supplier knew about this, but if he made reservations in his proposal, he ran a risk of being deemed 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. (In the final system, the traditional approach is used internally.)

Always getting data from the external system degrades availability and response time. If the external system is out of service, the Land Registry system will essentially be out of service too. A similar argument holds for response times.

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 Land Registry 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 over-night synchronization.

Not surprisingly, the final system has a local copy of all the external data with over-night synchronization of changes. The only exception is the digital signatures in DanID.

In spite of the promise, the external systems were not stable. All of these systems (except the civil registration system, CPR) were under major revision. Furthermore, all of the systems had to accommodate changes made specifically for the Land Registry system. These issues were very costly and time consuming to deal with for the supplier.

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, and it increased cost and time.

As another example, the customer (and his consultant) had specified that the system had to be available 99.8% of the time. It is easy to ask for this, but customers don't realize the cost. In the Land Registry case, the cost of operating the system with 99.5% availability is around 5 M DKK per year. A 99.9% availability costs around 15 M DKK per year. Is it worth it? The basic issue is that the customer may unknowingly ask for something that is much more costly than necessary.

This contributed with an added cost of 10 M DKK per year.

Effects on damages: Time, Cost.

Cause A5. Oversells technology

As explained under cause A4, SOA was costly and time consuming, thus contributing to the observed damages.

Effects on damages: Time, Cost.

Cause A7. Wants everything at once

The Land Registry system was deployed as a big bang with the entire country covered. The supplier had warned against it, but the customer insisted because citizens had to be treated equally. Otherwise it would be against the law. As it developed, the big bang caused a much larger inequality: 30% of the citizens got financial problems because recording had to be manual, which took 50 days instead of 10. Part of the 30% were records picked at random for checking.

A pilot operation, for instance for one of the 24 Danish court districts, would have allowed the parties to assess the load on hotline, the effect of usability issues, and the need for authorization of lawyers to register on behalf of their clients.

We have seen systems that tried to cover all the *complexities* of the domain (for instance EFI), but here the Land Registry was pragmatic. Complex registry cases were to be handled manually.

Effects on damages: Business results, User satisfaction.

Acquisition

No damage observations.

Design

Cause C1. Doesn't ensure usability, even when they know how

Usability means that users are able to use the system efficiently without someone to guide them. 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 supplier 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:

> *[We will test with] Rolf Molich's principles from his book . . .*

Molich is a Danish usability specialist and - like other usability specialists - he recommends thinking-aloud tests with early prototypes and only the guidance that would be available in real life. 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 is not a usability test but what developers often call a "user test". It doesn't ensure usability because in real life nobody is available for guiding the user and helping as needed.

Usability tests were not 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 (the land registry judge) and the supplier's graphical designer developed the user screens, but didn't do any usability testing. Usability experts know that a user interface designed in this way is understandable to a domain expert only. And this turned out to be the case in this project too. Even the lawyers and real-estate agents didn't understand the user interface and had to call hotline - which became overloaded.

In the National Auditor's interview with ten 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, however.

We have seen this confusion about user testing versus usability testing over and over. Even the Danish government body in this area (Digitaliseringsstyrelsen), requires user testing, probably in the belief that it means usability testing.

Effects on damages: Business results, User satisfaction.

Cause C2. Designs user screens too late

Usability specialists agree that it is important to make early prototypes (mockups) of the user screens, make usability tests of them, improve them and test again until the result is satisfactory. Usually two or three iterations suffice. Research shows that any programming made at this stage, will make it hard to improve the user interface because it seems too costly to throw away programs. Research also shows that if you follow the specialist advice, total development will be faster and cheaper.

It should be obvious that the Land Registry designed user screens far too late. The direct consequences were low usability and an overloaded hotline.

Effects on damages: Time, Cost, Business results, User satisfaction.

Cause C4. Cannot see how far the supplier is

The supplier had planned to build the system with a team led by two very experienced developers in Aarhus (Denmark). However, Google set up a department there and hired the two developers. This slowed down the project and the customer didn't notice.

It is not easy for a customer to see how far the supplier is. What should he ask for? Hours spent or technical descriptions? Don't say much. However, if the user screens were developed early - as recommended in cause C2 - he could see them and verify that they had passed the usability test. Later he could check progress by seeing how many of the screens worked.

Effects on damages: Time.

Programming

Cause D2. Surprises with system integration

System integration caused many problems. See cause A4.

Effects on damages: Time, Cost.

Test

No damage observations.

Deployment

Cause F1. Deploys the system with insufficient support or training

This was obviously an important cause of damage in the Land Registry.

Effects on damages: Business results, User satisfaction.

Cause F3. Wrong estimate of human performance

It turned out that registry staff worked much slower than expected. It was also a surprise that so many lawyers and real-estate agents asked for authorization to register on behalf of their clients. This was partly due to few citizens having got an electronic signature from the government, a process that was very hard for citizens with modest IT experience.

Effects on damages: Business results.

Management

Cause G2. Doesn't reschedule, but assumes the rest can be compressed

Many events on the way indicated that the project would be delayed, e.g. that the key supplier staff left early, the unexpected integration complexity, the delayed user interfaces, and several items in the risk analysis. As a result, registry staff left before the system was deployed, and temporary staff introduced registry errors. This could have been remedied with early rescheduling.

Effects on damages: Business results.

Cause G4. Doesn't face the danger, risk assessment downplays the danger

During the project the parties made regular risk analyses, but they were used mainly for arguing that the risk wasn't important. Most of the bad things that actually happened had been identified as risks, but no action was taken. As an example, we find these risks early 2007 (abbreviated):

ID	Risk	Level: 5 highest	Con-sequence	Status/comment
1	SOA is immature	1		Tax dept. 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 sys-tems	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. The risk analysis suggests that there is no problem.

Risk 2: The customer (the Danish Courts) had experience with IT systems for internal use, but had never 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. The risk analysis just downplays the danger.

Risk 3: As explained in cause C4, the supplier had planned to use a team led by two very experienced developers. However, they were bought by Google. The risk analysis suggests that it can be remedied by a "tight project management".

Risk 4: As explained above, the systems were not stable.

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

ID	Risk	Level: 5 highest	Con-sequence	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 the domain expert and the supplier's graphical designer. 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 or undesirable.

It should be obvious that the risk analysis was not made correctly. There were no safeguards and nobody took action for the high risks.

Effects on damages: Time, Cost, Business results, User satisfaction.

Cause G6. Cashes the benefit before it is harvested
The 220 registry staff knew they were going to be dismissed, but not quite when. Many of them left early. Later, the rest were dismissed with the standard notice of around 6 months, which was planned to be around three months after deployment of the system. But the schedule slipped once more, and these employees had left before the system was deployed. As a result, many registrations were made by temporary staff, who made many mistakes. After the big bang, many of these mistakes were revealed for the 30% of registrations that were handled manually. This caused further delays.

Effects on damages: Business results.

Cause G8. Excessive management or expert involvement
Often development is driven by domain experts or enthusiastic managers, but they cannot see the system from an ordinary user perspective. If they have a dominating influence, the result can be a system with low usability.

The land registry judge was a domain expert and had a dominating influence. He designed the user interface together with a graphical designer, and insisted on a law language that even lawyers and real-estate agents didn't understand. As an example, these users couldn't find out how to register a condominium deed. There were several options in the menu: Single-family housing, cooperative apartment, farm – but no condominiums. After weeks of waiting to get through to hotline, they learned that they had to select "Single-family housing". Land Registry staff had often heard this problem and suggested to add "condominium" to the menu, but the judge refused. The law said "Single-family housing" and so it be.

Effects on damages: Business results, User satisfaction.

Cause G11. Believes law blocks sound approaches
As explained in cause A7, the customer used as an excuse against running a pilot operation, that it was against the law. (Even if it had been against the law, the Land Registry could have asked for an exemption.)

Effects on damages: Business results, User satisfaction.

Cause G12. Insufficient staffing
The supplier had planned to build the system with a team led by two very experienced developers in Aarhus (Denmark). However, Google set up a department there and hired the two developers. This slowed down the project. See C4.

Effects on damages: Time.

Method for data collection
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 plus requirements and proposal as attachments).
4. Design specification, change specifications, risk assessments, etc. (more than 2000 pages).

We conducted interviews, sent drafts for review, etc.:

5. We interviewed real-estate staff and lawyer staff to learn about the system and the problems it caused. They also showed us how the system worked and how they used it.
6. 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.
7. 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.
8. 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.
9. We wrote our findings as a preliminary report and submitted it to the customer for review, discussed disagreements, and published the final report [8].
10. 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. As a result Lauesen published the scientific paper *Why the electronic land registry failed* [4].
11. As part of publishing this working paper (*Damage and damage causes*), Lauesen sent the first draft to the (now former) president of the Danish Courts. At that time Lauesen used the term "disaster", which is commonly used in this field. The former president was concerned that this gave a wrong impression, since the project actually became a success. Lauesen discussed several alternatives with colleagues and ended up using the term *Damage* instead. The former president was also concerned about the way costs were specified. Costs can be defined in many ways and Lauesen reverted to using the figures from the National Auditor's report.
12. In August 2011 a group of citizens and a group of companies charged the Danish Courts for losses caused by long delays in the Land Registry. The courts had given the parties a no-pay exemption, since it was a milestone case. The case was handled by the High Court.
13. In May 2019 the court ruled that the Land Registry was not accountable for the losses. The plaintiffs got permission to escalate the case to the Supreme Court, still without pay. As part of this, the plaintiff’s lawyer consulted Lauesen about the usability requirements, how usability was defined and how high-usability systems could be implemented in practice. As part of our discussion, the lawyer

explained that the land registry judge had admitted that some of the delays in the registration were caused by registration cases being picked at random for manual checking, which due to the overloaded office caused several weeks of delay. The land registry judge had denied this earlier.

14. The supreme court ruled that since usability testing was not common practice at the start of the project, damages caused by low usability was not the responsibility of the Land Registry.

2. Travel card, RK

In 2002 Denmark had around 20 public transport operators comprising bus, train and ferries. Each company had their own ticket systems and fare calculations. It was suggested to establish a shared electronic travel card system for the entire country. The company RK (Rejsekortet - Travel Card) was established in 2003. It had many of the transport operators as shareholders and DSB (Danish State Railroads) as the largest. In 2003 RK sent out a request for proposal. The delivery comprised card scanners in busses and on stations, driver screens in busses, cabling, servers, networks and software, as well as operation and maintenance for 10+ years. Software comprised software in the cards, scanners, driver screens, servers and back-office. In 2005 RK signed a contract with EW (East-West, owned by Thales) for pilot operation in 2007 and full operation in 2009. But many delays occurred and real operation started late 2011, full operation in 2013.

Below we will focus on the software part, including integration to other systems. In a few places we may mention other issues.

Observed damages

a. **Time**: The development time was estimated to 3.5 years. It became 7.5 years.
b. **Cost**: Hardware and cabling was 500 M DKK. The estimated SW cost was 585 M DKK. It became 600 M DKK. We do not consider this a damage. The internal costs until deployment were around 100 M DKK. This was much higher than planned, due to the long development time.
c. **Business results**: The system operates successfully and gives the planned modest income. It was expected that the transport operators could save costs by phasing out their old ticket systems and optimizing time tables. Little of this has taken place at the time of writing (Feb. 2017).
d. **User satisfaction**: Initially users were dissatisfied with usability of the back office systems (buying cards, changing the related bank account, etc.), strange fare calculations (different for travelling one way or the other, rules differed from paper tickets, etc.).
e. **Supplier loss**: The supplier probably lost 500+ M DKK, primarily because he hadn't considered the cost of back office.

State today

The system is heavily used. Usability problems have gradually been removed and people get used to the rest. Travelers say they travel more when they have the card.

References

1. Rigsrevisionen (National Auditors): Beretning til Statsrevisorerne om rejsekortprojektet (Report to the State Auditors on the Travel Card project), 47 pages (in Danish) (June 2011).

Analysis

Cause A1. Doesn't identify user needs and win-win

The customer (RK) didn't know how to operate such a system. Which back office and web features had to be provided? How would travelers buy the card on the web and later fill it, how would RK monitor the operation, how would accounting work, how

would fare rates be changed, etc. They assumed that the supplier delivered all of this and didn't study how back office worked in cities where Thales operated, e.g. Amsterdam.

It turned out that the supplier had never delivered back office systems. They provided technical interfaces (API's) to their system that could return data on travels per card, record paid amounts on the card, block stolen cards, etc. The back-office systems were developed by local transport operators. These systems accessed card data by means of the technical interfaces. The supplier expected RK to develop back office systems in the same way.

Effects on damages: Time, Internal costs, Supplier loss.

Cause A2. Requirements don’t cover customer needs

The customer's requirements specification was written by many independent engineers, each of whom covered a narrow technical area. The result was 60 files each around 15 pages, all starting with requirement 1.

Here, we look at the SW issues only, particularly the back office. The requirements used the style recommended by IEEE 830 (American engineering standard for requirements, 1993): Specify the functions to be provided by the system. Here is an example from the RK case:

K30 *There shall be facilities for reporting fraud. The reports shall show the number of times fraud was registered for each of the types of fraud known in the System.*

K 119 *Being able to answer questions from customers. Supplying information about products, prices, conditions, etc. (not timetables).*

This kind of requirements fails to address the basic question: Who is going to use these functions, when and for what? To answer this question, it is necessary to know what back office is supposed to do and what travelers have to do through the web. As explained for cause A1, neither supplier, nor customer had this knowledge.

The supplier tried to argue that the requirements didn't specify that he had to develop back office support and web site, but he ended up accepting that it was his responsibility and covered the costs himself. He contracted with the Danish company Accenture to develop it.

The requirements were more successful about usability. They specified that early mockups of the screens had to be made and usability tested. However, they didn't specify that the supplier had to remedy the usability problems. This is taken for granted in a Danish context, but not internationally. It was further complicated by conflicts between supplier and the subcontractor.

Effects on damages: Time, Internal costs, User satisfaction, Supplier loss.

Cause A8. Doesn't plan the new work processes

As explained for cause A1 and A2, the customer didn't know how to run a back office, etc. He believed the supplier knew.

Effects on damages: Time, Internal costs, User satisfaction, Supplier loss.

Cause A10. Surprising rule complexity

The basic idea with the travel card was to pay for the end-to-end geographical distance. However, this was not acceptable to the Danish transport operators. They divided their districts into zones, and travelers paid for the number of zones they visited. Fare rules also varied, for instance whether you could break the journey and for how long. Further, trains and busses could serve the same district, but have different zones.

With a different fare system, some operators would earn more than today, others would go bankrupt. And it was impossible to tell in advance who would do what.

These issues ended up with a complex set of fare rules: 65 pages of rules and 100 pages with examples of how to compute the fare. Amazingly, the supplier handled all of this without problems. This was his business - not the back office.

Effects on damages: None

Acquisition

Cause B1. Supplier too optimistic - you must lie to win

You must lie to win is a quotation from the supplier's CEO, but many suppliers recognize it. They hope they can sort it out after the contract, for instance by claiming that something is a change request rather than a defect.

The supplier probably saw the risk with the back office, but wanted to give a price low enough to get the contract. How can the customer deal with this? He can ask for an early proof of concept (POC), in this case a mockup of the back office screens. If the POC fails, the customer can terminate the contract and select another supplier. Honest suppliers favor such a rule in the contract, because it allows them to win when a too optimistic supplier fails the POC.

Effects on damages: Time, Internal costs, Supplier loss.

Cause B2. Wrong selection criteria

The proposal was hard to assess. As an example, the reply to K30 above about reporting fraud was this:
Reply: Operator through a dedicated user interface enters the criteria used for report generation.
How would this work in practice?

In many cases the requirement was obscure. The supplier replied with a slight reformulation of the requirement. Here is an example:
Req 276: Presentation of information on the screens must be clear and stable.
Reply: Screens will be designed in such a manner that information is clear and stable.

Although the supplier had delivered solutions to many cities, the customer didn’t assess what the supplier had actually delivered. So selection was done on the price only.

Cause B3. Wrong cost estimates

The supplier more or less consciously forgot the back office system. Even when he accepted to deliver it, he much underestimated the effort needed.

Effects on damages: Time, Internal costs, Supplier loss.

Design

Cause C2. Designs user screens too late

The user screens should have been part of the solution description (*System Specification*). They were not, and this was a strong indication that the supplier didn't know what to do about the back office. See also C3.

Effects on damages: Time, Internal costs, Supplier loss.

Cause C3. Accepts the solution description without understanding it

In 2006 the supplier submitted a solution description (*System Specification*). It comprised 248 files, but was mainly a description of cables, servers and other hardware. There was very little on software, and what was written was hard to understand. As an example, there was a 500 page data model with names of all fields in the database, but nothing about what the fields were used for. There were a few user screens. They showed the list of hardware components and their state as it appeared to the technical maintenance staff.

The customer didn't understand all of this, and didn't know what to expect. His reaction was: EW has promised to deliver, so we wait and see. As a result, the customer accepted the description with a few comments on some detail.

There were several early warnings that something was all wrong, but management ignored them (see G4).

In 2007 the supplier (and Accenture) developed the back office system. According to the contract, they ran usability tests. The tests showed lots of usability problems, but the supplier refused to do anything about them. The contract didn't explicitly require him to do it. It was further complicated by conflicts between supplier and the subcontractor.

In 2008 a pilot test was made in West Zealand with 50 travelers who used the system for free. The back office system turned out to be of little use. In 2009 accountants refused to accept the system. There was no audit trail, so they couldn't relate the payments to the travel transactions. The supplier pointed out that nothing was stated about this in the requirements. The customer referred to a requirement that said that the system shall comply with all Danish laws and regulations, and this included the law on accounting, which required audit trails.

In 2010 the parties wildly disagreed on what to do. They tried use cases, but soon agreed that they didn't work. (It was the kind of use cases that describe what the system does, technically speaking.)

They ended up agreeing to make a task force consisting of two experts from the supplier, one from DSB (the main stakeholder in RK), one from Movia (a bus operator) and one from RK. They got full authority to do what they found necessary.

In 2011 the back office and web were ready and had been usability tested. True operation of the system started end of 2011.

Effects on damages: Time, Internal costs, Supplier loss.

Cause C4. Cannot see how far the supplier is

See cause C3.

Effects on damages: Time, Internal costs, Supplier loss.

Programming

Cause D1. Supplier accepts the expensive requirements interpretation

Considering the very obscure customer requirements, it is remarkable that the supplier accepted to cover the full costs of the back office system himself. One reason he accepted, is probably that he wanted to prove that Thales could serve a whole country with many transport operators. Denmark would be a showpiece in that direction.

Effects on damages: Supplier loss.

Test

No causes.

Deployment

No causes.

Management

Cause G1. No business goals - or forgets them on the way

The business goals were quite weak:

a. The travel card would give 2% more travels. If the card got only one percent more travels, the business case would be negative. 2% is not measurable in light of all the other changes that take place in society. However, in questionnaires travelers say that they travel more when they have the card. Earlier some transport operators had observed a similar behavior with other ticket media. So we trust that this goal is met.
b. Transport operators could save costs by phasing out other ticket systems. At the time of writing (March 2017), very little has been done in this area.
c. Time tables could be optimized based on data from the travel cards. The author doesn't know how much has happened in this direction.

Points b and c are not the responsibility of RK but of their stakeholders. However, in the view of society, it is questionable to spend so much money without a measurable benefit.

Effects on damages: Business results.

Cause G4. Doesn't face the danger, risk assessment downplays the danger
Most of the risks the author has seen where about internal details or reports being late. However an early risk assessment said that the experience from Holland was that Thales couldn't document systems and gave unrealistic promises. This turned out to be true in Denmark too. The customer didn't face the danger. He should have used the risk assessment to check the customer's proposal better and make an early proof of concept, including documentation.

At the time of accepting the solution description (C3), the customer's IT specialists warned management that something was all wrong. This too was ignored.

Effects on damages: Time, Internal costs, Supplier loss.

Cause G5. The financial incentive disappears and the parties fight instead of cooperate
This is what happened in the long period where the parties didn't agree (2007 to 2010). There were many contract additions in this period, but they were about new deadlines and payments. Nobody asked what the basic problem was.

Fighting ended when a small task force was given authority to do whatever they found necessary (see cause C3). Later a Dispute Resolution Board with three external persons was established to mediate in conflicts.

Effects on damages: Time, Internal costs, Supplier loss.

Cause G9. Too large steering committees/working groups without competencies
RK was based on the idea that the supplier shouldn't discuss with all the transport operators, but with RK only. However, in practice all operators participated in meetings and it was very hard to agree on anything. As the supplier's CEO said:

In Denmark everybody can say no, but nobody can say yes.

Real progress started when a small task force was given authority to do whatever they found necessary (see cause C3).

Effects on damages: Time, Internal costs, Supplier loss.

Cause G13. Doesn’t find the root cause
As explained under G5, there was a long period with fights about new deadlines and payments (2007 to 2010). Nobody asked what the basic problem was.

Method for data collection
(Not available)

3. Police case management, PolSag

The Danish police had an old, partly paper-based system to keep track of offences and related documents. Each police district had their own archives. They decided to acquire a more modern system to be shared between all 12 districts. After 6 years, the new system was deployed in a pilot test. It was not successful and soon after the project was closed.

Observed damages

a. **Time**: The development time was estimated to 3.7 years from project funding (Jan. 2005). The project was closed in 2012 after 7 years.
b. **Cost**: The estimated software cost in 2005 was 173 M DKK. When the project was closed, 354 M DKK had been spent on software, 68 on operation and 145 on internal costs (salaries). Much more investment would be needed to complete the system.
c. **Business results**: Negative. Nobody could explain what the benefits would be, but the yearly operating costs would be 5 times the present ones.
d. **Supplier loss**: The subcontractor lost 50+ M DKK.
e. **Feasibility doubts**: Performance was never proved, the system seemed faulty and with bad code quality.

State today

Project closed. The supplier paid 160 M DKK in compensation. The police (RP) still don't know what to do about case management.

References

1. Rigsrevisionen (National Auditors): Beretning til Statsrevisorerne om politiets it-system POLSAG (Report to the State Auditors on the police IT system POLSAG), 56 pages (in Danish) (March 2013).

Analysis

Cause A1. Doesn't identify user needs and win-win

The plan was to use a commercial case management and document handling system (ESDH) and extend it a bit to deal with special police issues. Captia from Scanjour was selected. The present supplier (CSC) became the main contractor with Scanjour as a subcontractor. In order to support the police, Scanjour would look at the present work processes of the police to see how they could be supported with Captia. This was the way Scanjour normally worked.

At the first workshop with the Police, 40 policemen turned up. They couldn't tell what their work processes were. They were confidential, they said. As the Police admits today, they didn't know their work processes and had never tried to describe them. The confidentiality was just an excuse. The supplier's developers had been security cleared, so confidentiality wasn't an issue anyway.

In this situation, the parties tried to figure out how the existing user screens worked and then make similar ones in Captia. There were 100 screens in the present system, but nobody quite knew what they did. They had developed over time and much of the old supplier's staff had retired. Much of the functionality was quite complex, e.g.

automatically dealing with changes when a suspect was believed to be above 18, but turned out to be 17 at the time of the crime, and as a result had to be handled according to different rules.

So the screens were to a large extent reinvented. During this, the police came up with many things that also would be nice to have. It ended up with around 210 screens plus 500 database tables to be added to Captia's 300 tables, e.g. dealing with automatic speed trap data (ATKS), the list of football banned citizens, the list of animal transports to be checked, etc.

Effects on damages: Time, Cost, Business results, Supplier loss, Feasibility doubts.

Cause A2. Requirements don't cover customer needs

Initially there were no real requirements. The early decision was to buy a commercial case management system that the government already had contracted on, and then work out the requirements in cooperation with the supplier.

The requirements didn't specify who would use the system, when and for what. Instead there were long lists of functionality, e.g. search criteria to be supported. Further, this requirement tried to cover the rest (called the 1:1 requirement):

> *All existing functionality [in the old system] must be provided in a user-friendly way in the new system. Providing similar functionality means establishing functionality - based on the Captia-solution [the supplier's proposal] - that supports the same business processes and solves the same problems as the existing system does today.*

This would be a useful requirement if the existing work processes and problems had been specified, but as explained for A1, this was not possible.

As a result, the requirements later became "design-level requirements" that specified the technical solution in detail, e.g. what each button on the screens would do. This was done as "use cases" that described details of the screens and how they interacted with other parts of the system. This means that the real user needs were not visible, so it was impossible to see what could have been done with the built-in screens.

In this world of technical specification, usability is not an issue and no usability requirements where specified.

Effects on damages: Time, Cost, Business results.

Cause A3. Describes the solution in detail. No freedom to the supplier

As explained in cause A2, the requirements became very detailed (technical level). So the supplier had little freedom. It helped that the supplier was part of the design team, but he had no way to twist the work processes so that built-in screens would suffice.

Effects on damages: Time, Cost, Supplier loss.

Cause A5. Oversells technology

The main technology in this project was *case and document handling systems* (ESDH). The government applied such systems in many departments, and they considered police cases just more of the same. However, the police had so many special tables (500, see cause A1), that it was more of an ERP system than a case handling system.

SOA was promoted in this project too. The Police had several feeding systems (Randsystemer) that provided structured data, e.g. speed trap data (ATKS). The idea was to add SOA services to them, but it ended up as costly additions to all of them. ATKS was changed to run in two versions, one with and one without SOA.

Total cost of changing the feeding systems was 55 M DKK.

Effects on damages: Time, Cost, Supplier loss, Feasibility doubts.

Cause A7. Wants everything at once

The Police wanted to include all feeding systems, instead of starting with plain case and document handling (ESDH) and then adding feeding systems when there was a positive business case. They had planned to deploy the system district by district, which was fine.

Effects on damages: Time, Cost, Business results.

Cause A8. Doesn't plan the new work processes

The Police didn't know their present work processes, and planning new ones were beyond the horizon. This also meant that nobody cared about the expected benefits.

Effects on damages: Time, Cost, Business results.

Cause A9. No feasible solution

Consultants involved in the first stages of the project, had warned that Captia had never handled that many users at the same time, and that the very architecture of Capture might be a bottleneck causing long response times. They strongly suggested to make a feasibility test before starting the project. This feasibility test was never made.

In the last months before the project was closed, performance testing was attempted. At this point in time, money ran out and the parties fought instead of cooperate. CSC made their test, Scanjour their, and they didn't agree on the results. Globeteam was called in as an external consultant and made one measurement. It showed an incredible number of database calls. Everybody could see that this was all wrong, but nobody had the time to look at it. Scanjour suggested that Globeteam might have used a test version that logged every little system action in the database. This would explain the huge figures. A whistleblower told us that the Police had forbidden Globeteam to clarify the issue, because the lawyers wanted to prove a breach-of-contract.

Effects on damages: Feasibility doubts.

Acquisition

Cause B3. Wrong cost estimates

Scanjour had estimated their price based on the 100 existing screens that had to be redesigned in the Scanjour way. They ended up with 210 screens. They couldn't see how much special functionality was behind each screen. Also the high costs of SOA were a surprise.

The customer forgot to include the cost of operating the new system during development (68 M DKK), the cost of training the users, and the cost of operation and maintenance after deployment. The contract stated that the yearly fee for maintaining the system was 25% of the entire development cost. So lifetime costs grew wildly as development costs grew.

Effects on damages: Time, Cost, Business results, Supplier loss.

Design

Cause C5. My way without considering the supplier's way

As explained under cause A1, it was impossible to see whether the supplier's user screens were sufficient, maybe with small changes.

Effects on damages: Time, Cost.

Programming

Cause D1. Supplier accepts the expensive requirements interpretation

Scanjour management didn't notice the growing costs during development. When 80% of the funds were used, they realized that only 40% of the work had been done. They told us that they didn't understand how this could happen. They accepted to cover the additional costs themselves.

A better understanding of how the project had changed, might have given them arguments for renegotiating the contract. And they should of course have done so early in the project.

Effects on damages: Supplier loss.

Cause D2. Surprises with system integration

System integration caused many problems. See cause A5.

Effects on damages: Time, Cost, Feasibility doubts.

Test

No damage observations. During testing, a lot of problems were detected, but most of them were easy to fix, e.g. many misprints in the forms that the police used for sending documents to the courts. The root problem was not difficulty of testing, but the parties not agreeing on who should detect and repair the defects. This is covered by cause G5, the financial incentive disappearing.

Deployment

Cause F1. Deploys the system with insufficient support or training

There was a pilot test in a small, remote police district in Denmark, Bornholm. CSC was in charge of the deployment, and the subcontractor (Scanjour) wasn't directly involved. A lot of problems came up and rumors were that the system was slow, faulty, stopped working, etc. Apparently bad code quality.

However, many problems were not bad code, but wrong configuration, the customer's insistence on harmful requirements, etc. As an example, the system took two minutes to log on. The reason was that the Police had provided a list of five URL-addresses for user directories (AD's). The system had to try them one by one until it found the user. But the first AD didn't exist. The system waited 30 seconds, then tried the next - which also didn't exist. Only the last one existed.

CSC thought Scanjour's system was slow, but didn't inform Scanjour. When Scanjour heard about the problems weeks later, they located the problem immediately, but were not allowed to change anything.

Effects on damages: Feasibility doubts.

Cause F2. Doesn't check whether the system is used as intended

At the pilot test nobody checked that the system was used as intended. As a result, rumors spread about low technical quality, when it actually was user misunderstandings.

Effects on damages: Feasibility doubts.

Management

Cause G1. No business goals - or forgets them on the way

The Police had some old goals about benefits and cost saving, but when the National Auditors investigated the project, nobody knew the goals. Nobody had paid attention to them during the project. Supplier as well as subcontractor had asked the customer what the purpose of the system was, but never got an answer.

Further, nobody had paid attention to the growing cost of maintenance (see cause B3). As a result, nobody could come up with a reason to continue the project.

Effects on damages: Business results.

Cause G3. The project grows without anybody noticing

As explained under cause D1, Scanjour management didn't notice the growing costs.

Effects on damages: Time, Cost, Supplier loss, Feasibility doubts.

Cause G5. The financial incentive disappears and parties fight instead of cooperate

The reason Scanjour didn't support the Bornholm pilot test was that money had run out. It was also the reason the performance issue and other issues weren't settled.

Effects on damages: Time, Feasibility doubts.

Cause G7. Lack of management involvement
The Police changed project management several times, and for a long period had external consultants as managers. Apparently the managers didn't understand the real problems and didn't know what to do.

Effects on damages: Business results, Feasibility doubts.

Cause G10. Excessive user involvement
Having 40 policemen participate in the initial analysis, didn't work out. However, this soon stopped.

Effects on damages: None.

Cause G13. Doesn’t find the root cause
As explained under A9 and F1, there were many rumors about errors and low performance, but the parties didn’t look for the causes.

Method for data collection
(Not available)

4. Debt collection, EFI

When citizens don't pay their tax or other debts to authorities, the bailiff (Danish: pantefoged) will take action. He can hold back part of the debtor's salary, sell his property, arrange a payment scheme, etc.

In Denmark each authority had their own bailiffs and ways of collecting the debt. Authorities included the tax department, the state railroads (travelling without a valid ticket), electricity suppliers (not paying the electricity bill), municipalities (not paying child allowance), public schools (not paying the fine for hitting a window with the football), etc. In total around 12 million debts are recorded every year, a surprising number considering that Denmark has barely 6 million citizens.

The government decided that all debt collection for municipalities and the government had to be centralized and handled by the tax authorities, who handled the largest amount of debts anyway. Initially (2005) all bailiffs and other specialists were organizationally moved to the tax department.

The idea was that a new IT system (EFI) would automate debt collection and spare around 300 jobs. EFI could through a datawarehouse get data from many other systems to find out whether to hold back salary, sell property, etc. Actually, it was a great idea. However, EFI was delayed. The result was that debt collection decreased and the total amount of debt increased.

EFI had to be part of the SOA architecture that the tax IT department (*Tax*) had created over several years. Only five of the hundreds of tax systems had been integrated by SOA and it had been very costly (see A5 below). The plan was that the benefit would be harvested with EFI. However, it turned out that debt collection was much more complex than anticipated and the rules built into EFI sometimes violated the law. Further, authorities didn’t report debts correctly to EFI, and EFI didn't check the data it got.

Observed damages

a. **Time**: The development time was estimated to 2.5 years. Actually, the first part of the system was deployed after 8.5 years (2013).
b. **Cost**: The estimated project cost was 144 M DKK. The actual cost was around 600 M DKK when the first part of the system was deployed.
c. **Business results**: The plan was that the system should save salaries of 300 bailiffs and speed up debt collection. The bailiffs were actually dismissed, saving around 150 M DKK per year. However, it turned out that most debts could not be collected automatically, either because data were wrong or missing, or because human judgment was needed (ref. 1, page 48). Automatic debt collection was soon stopped, becoming partly manual. Around 300 case managers were employed to do this, eliminating the gain of saving the 300 bailiffs. Collection of 70,000 M DKK debt was delayed for years.
d. **Feasibility doubts**: In September 2015 the feasibility of the IT architecture was questioned. Accenture wrote a report that clearly said that the programs involved couldn't be rescued (see A6, multi-vendor strategy). Further the State Lawyer (Kammeradvokaten) declared that the system in many ways violated the law.

Actually the system did only what human debt collectors had done before (see G11, believes law blocks sound approaches).

State today
The system was deployed September 2013. There were many errors in EFI and related systems, but they were gradually removed. The system automatically collected debts, e.g. by holding back part of the debtor's salary. However, when the architecture and legality were questioned in September 2015, Tax stopped the automatic debt collection.

At this point in time, there were 20 M debts, totaling 74,000 M DKK. According to ref. 1, page 48, it would require 12-25,000 person years to sort out what to do with the dubious debts. This was clearly unacceptable. Tax took several measures to deal with the situation: The automatic expiration of debts was put on halt by law. Around 80% of debts couldn't be collected because the debtor was too poor. Tax/Parliament cancelled much of it or let it expire.

Case managers were to continue manual debt collection, supported by EFI. At the time of writing (June 2017) the logs show that 1600 EFI users do this. However, due to the detected legality issues, the most efficient collection method - holding back salary - is not even used manually. The system still supports the process, e.g. recording 12 M new debts and generating around 30 M notes and letters per year. The press and general public believed that the system had been closed down completely.

Tax has contracted with another supplier to develop a new system based on an ERP system that supports collection of public debts. It will gradually take over the work of EFI. The legality issues have been settled and complexity reduced (e.g. 5 types of debt rather than 400+ types). The system checks debt reports for correctness and completeness, and expert users are involved a lot in the design process. All of this is great. It is a pity that it wasn't done a decade ago in EFI.

References
1. Skatteministeriet (Ministry of Tax), September 2015: Redegørelse om ét fælles inddrivelsessystem, 58 pages (in Danish).
2. Rigsrevisionen (National Auditors): Beretning til Statsrevisorerne om SKATs systemmodernisering (Report to the State Auditors on Tax’s system renovation), 34 pages (in Danish) (January 2015).
3. Accenture Consulting, September 2015: Overlay Report, Assistance for analysis of "Et fælles inddrivelsessystem", 24 pages (in English).

Analysis

Cause A1. Doesn't identify user needs and win-win
Management had a very simplified picture of how debt collection worked. *It was all much of the same, maybe with a few variations*. However, bailiffs had many tricks and concerns in their duty, unknown to management. Examples:
1. A good approach with an arrogant debtor is to locate him at his favorite bar where he has drinks with his fellows. While they all listen, announce his debt and ask whether he cannot pay or whether they should set up a payment scheme. Suddenly, he is willing to pay.

2. A social client and alcoholic is back at normal and has got a job. This is not the time to collect missing child allowances. Give him a year. The authorities often did such things.
3. A family has suddenly got an electricity bill that is twice as large as usual. They cannot pay. The bailiff visits them and finds out that they have installed infrared heating on the balcony. To their surprise, it consumes as much power as the rest of the household. They agree on a payment scheme.

Computerized debt collection cannot do things like these. Tax explains that this can still be done by authorities before reporting the debt to EFI. However, in practice they don't because they don't have the expertise anymore.

Another issue was that existing data on debts had poor quality, as explained above. As long as debt collection was manual, this wasn't a problem. But when it was automated, it became a big problem.

Effects on damages: Time, Cost, Business results.

Cause A2. Requirements don't cover customer needs

Requirements were written as use cases and service definitions. There was nothing about how to deal with the many business rules and laws. Further, the requirements failed to address the basic questions: who is going to use this system? when? and for what?

Effects on damages: Time, Cost, Business results.

Cause A5. Oversells technology

In 2004 Tax decided to base 6 new systems on a service-oriented architecture (SOA). The first system was the integration platform itself. The last system was EFI, and the main benefit of SOA was to be harvested here. At that time SOA was praised by many consultants and recommended by the government. It sounded easy: We can just develop the systems we want and then connect them by means of services.

It worked, but was very cumbersome and expensive. Development took 6 years longer than planned. The estimated cost for all six projects were 500 M DKK. The actual cost became 1,500 M DKK. The integration platform itself cost 120 M DKK.

Developing a new system required that you figured out which systems contained the data you needed. Next, you had to negotiate services for exchanging data, and when you had made a mistake renegotiate the service. Typically the cost for a service was 100-200,000 DKK.

The promises were that you could define the services up front in a logical and business-oriented fashion and reuse them in many projects. This wasn't the case in practice. The integration platform got 1,600 services. Around 100 of them were reused once and 20 of them several times. The rest were essentially point-to-point connections. Further, performance was low and availability vulnerable as a system would seem down when one of the systems it depended on was down. To provide sufficient availability, systems replicated data that belonged to other systems.

EFI should use data from other systems than the 5 new ones to find ways to collect debts, e.g. land registry, car registry, banks and salary systems. It was done through a data warehouse that all the systems reported to. This was relatively painless. However, EFI had to integrate with around 20 other systems and this caused the usual SOA problems.

Effects on damages: Time, Cost, Feasibility doubts.

Cause A6. Multi-vendor strategy - makes us supplier independent

With old IT systems, the government and the municipalities had experienced that the supplier essentially had a monopoly. The customer's new systems couldn't access existing data and only the original supplier could modify or extend the system.

SOA was one of the answers. Another was to use several suppliers. The customer imagined that if he wasn't satisfied with one of the suppliers, he could buy the system somewhere else and "plug it in". This was never done in practice. Instead the customer experienced that now he was fighting several monopolies instead of one.

In the case of EFI, Tax had divided claims management between two systems, one that recorded the tax payer's financial transactions with Skat (DMI), and one that supported the collection process (EFI). Each of them kept track of data in its own way, but had to integrate with the other. DMI was developed by CSC and EFI by KMD, two companies in fierce competition and with different development methods. This was not a good base for agreeing on services - nor on testing.

Furthermore, the logic behind debt collection was extremely complex and had to be split between the two suppliers. At the time of deployment, this had been tested in a few simple cases only. Many errors were detected and repaired later.

Tax had defined the services and components involved. The result was shown in an overview diagram with around 100 boxes and connection lines all over in "spaghetti" style.

In 2015 Accenture wrote a report that clearly stated that the programs involved couldn't be rescued. KMD has later explained that there were many problems right after deployment, but nearly all of them had been repaired when Accenture wrote the report. This is supported by the fact that the system worked adequately in 2015, apart from some legality issues.

Effects on damages: Time, Cost, Feasibility doubts.

Cause A7. Wants everything at once

Analysts knew that there were several types of debt. As an example, the rules and collection methods for tax debts were different from collecting traffic fines. Probably there were 10-20 different types of debt? However, closer analysis revealed around 500 types. Bending rules and practice a bit, brought the number down to 400 types, each with their own rules (see more in A10). The number later grew to 490.

Instead of starting with a few debt types that could collect a large amount, Tax wanted to cover all types. It would look unfair if some types of debt were not collected, and

manual collection wasn't possible since bailiffs had been dismissed. The result was a system that was extremely hard to test, amplified by the multi-vendor strategy (A6).

Effects on damages: Time, Cost, Business results.

Cause A9. No feasible solution

During analysis, developers had looked only at simple flows for debt collection. Was it possible at all to specify the entire logic, e.g. when a debt had several debtors, or was outdated and the debtor paid it anyway, or claims changed as a result of court cases?

Tax had experts in these areas, but they wouldn't cooperate and management didn't interfere. The result was that when the system was deployed, the entire logic turned out to be incomplete. Testing was incomplete, but couldn't have helped anyway because developers didn't know what to test against. Requirements were missing.

Another issue was whether it was possible and realistic to transfer debts from old manual debt files to EFI. This was not carefully investigated and the result was that most debts needed some manual check or repair before being handled automatically (see F2).

Effects on damages: Time, Cost, Business results.

Cause A10. Surprising rule complexity

To map some of the complexity, Tax analysts identified 700 rules that could apply for a specific type of debt. For instance: could the debt be collected from the spouse instead of the debtor? Would the collected amount go to the state or to a specific authority, e.g. a school or a transport provider? When would the debt be outdated?

Analysts set up a spreadsheet with 490 rows, one for each type of debt, and 700 columns, one for each rule. In each cell, they tagged whether this rule applied for this type of debt. This spreadsheet was later split into several spreadsheets and interpreted by programs that carried out the rules as specified. This was a clever approach and it put much of the burden of testing on the tax and debt experts.

However, there were many other rules that the spreadsheet didn't cover. One example was the rule about salary withdrawal that made Tax close down automatic debt collection in 2015 and announce the whole system dead (see G11).

Effects on damages: Time, Cost, Business results, Feasibility doubts.

Acquisition

Cause B3. Wrong cost estimates

The cost estimates were a factor 4 too optimistic. This is related to lack of true requirements (A2), optimistic expectations about SOA (A5), multi-vendor strategy (A6) and surprising rule complexity (A10).

Effects on damages: Time, Cost.

Design

No damage observations.

Programming

Cause D2. Surprises with system integration

As explained in A5 and A6, there were many problems with system integration.

Effects on damages: Time, Cost, Business results.

Test

Cause E1. Deploys the system with insufficient testing

The first part of the system was deployed September 2013 under heavy pressure from management. It had cost so much already and more was necessary to complete it. The suppliers had tested EFI and DMI in isolation, but cooperation of the two systems had been tested in only a few simple cases.

The checkmarks in the huge spreadsheets had not been tested, and this was Tax's duty. They planned 9,700 test cases. 5,700 of them passed, 3,400 failed and 600 hadn't been tried. Management erroneously concluded that half of the system was correct and could be deployed. Nobody investigated the test coverage, i.e. whether the 9,700 test cases covered all parts of the code and the 343,000 cells in the spreadsheet. Developers tried to explain, but management wouldn't listen and later claimed that they hadn't been informed.

There was a pilot test to check whether deployment would be acceptable, but it didn't succeed. Nevertheless, Tax decided to deploy the system. There were many errors at deployment, but they were detected and removed over the next years. Rumors about them persisted, however.

Effects on damages: Business results.

Deployment

Cause F2. The system is not used as intended

EFI received debts to collect from external claimants, such as municipalities and transport operators. Tax insisted that it was the claimant's own responsibility to select the proper debt type (out of the 490 possible) and supply the necessary dates and documentation.

The result was that many debt records couldn't be used for automatic debt collection. It also happened that bailiffs met with debtors, but were unable to document the debts.

When the system was deployed in 2013, all debts from existing systems were converted to EFI/DMI format. The system checked only one thing: That there was a limitation date (Danish: Forældelsesfrist). When it was blank, the system had to set it to April 1st, 2014. Tax imagined that this gave them time to find the correct limitation date, but later forgot about it. The result was that when April 1st, 2014, arrived, the system cancelled all these debts.

Effects on damages: Business results.

Management

Cause G1. No business goals - or forgets them on the way

- The business goal for EFI was weak: *Efficient debt collection*. There was nothing about how this would be measured.
- The goal for the SOA platform was this: *Purchase and deploy a new integration platform that for instance will improve communication between Tax's IT systems.*

There was no indication of how to measure and reach these goals, apart from dismissing bailiffs. During the project, Tax didn't monitor the goals or change estimates, except cost estimates.

Effects on damages: Time, Cost, Business results.

Cause G4. Doesn't face the danger. The risk assessment downplays the danger

Developers report that there was little communication between IT staff and management. The only point of contact was the project manager, who soon became absorbed by the political game in upper management.

Apparently management didn't understand the real problems behind the continuous requests for additional funding. Much was blamed on the suppliers, but their job was hard with the missing requirements and the IT architecture that Tax dictated.

Management didn't interfere when needed, for instance when expert help from lawyers and debt collection specialists was needed. When deployment was decided, management ignored the warnings about missing tests and high risks.

Effects on damages: Time, Cost, Business results, Feasibility doubts.

Cause G6. Cashes the benefit before it is harvested

Bailiffs were dismissed before it was sure that the project was feasible.

Effects on damages: Business results.

Cause G7. Lack of management involvement

There was little communication between IT staff and management. See G4 (doesn't face the danger).

Effects on damages: Time, Cost, Business results.

Cause G11. Believes law blocks sound approaches

The customer closed down the automatic debt collection in 2015 and announced the whole system dead. The primary cause was that the State Lawyer found some of the collection procedures dubious or against the law. One example is that in order to calculate the amount that can be withdrawn in salary without ruining the family, the authority must use the most recent salary data. The customer (Tax) had manually used the smallest of the previous two month's salary, and the EFI system did the same.

However, the State lawyer hinted that this was dubious and more recent salary data might hide in the systems. He mentioned other cases that were dubious too. Nobody tried to find other solutions, e.g. asking for clarification or law changes.

In the new system (see *State today* above), legislation has been settled and rules simplified.

Effects on damages: Business results, Feasibility doubts.

Method for data collection
(Not available)

5. Health record system, EPIC

The hospitals in the Copenhagen and Zealand regions comprise 17 hospitals and 40,000 healthcare workers. (Some of the hospitals are geographically separate, but organizationally united.) They handle around 4 million out-patient visits per year and 0.6 million in-patients. They used around 30 different systems for health recording and treatment. Patient visits were cumbersome because the clinicians had to find paper folders and log in and out of several systems. Usually the doctor dictated text for the patient record and a secretary later typed it into the electronic patient record and updated compulsory government systems, e.g. for settling of accounts. This was convenient for the doctor, but the patient record and government system were often 2-3 weeks out of date.

The regions decided to replace most of the old systems with one new. Five suppliers were pre-qualified and three sent a proposal. Each system was tested in a trial hospital department by three teams of doctors, nurses and secretaries. Each team got one day of training in the new system, followed by two days of practice. EPIC was clearly better than the competitors (score 7.9 vs. 5.9 and 4.8). The systems were also evaluated by 450 clinicians who saw the suppliers' sales demonstrations. EPIC was number two here, but the differences were small (the best got 7.83, EPIC 7.15). Based on this, many other factors and to some extent also the cost of the system (see B2 below), the regions chose EPIC and signed the contract in 2013. They decided to move part of the secretaries' work to the doctors, which would ensure that the patient record was always up to date and that there were no misunderstandings between clinicians and secretaries. It would also eliminate around 1/5 of the secretaries. Work with the new system should be much faster for the clinicians, since it wasn't necessary to look at papers and log in and out of many systems. It was also expected that doctors could record notes with a few clicks on predefined phrases.

EPIC comes with support for many medical specialties, but usually customization is needed. Integration with other systems is also special for each customer. Most of the around 20 medical specialties needed customized user screens, for instance with standard treatments. It was planned to let clinicians develop them after some training and with inspiration from the hundreds of special screens developed by other EPIC customers.

The system was deployed in the first two hospitals mid 2016, the next 6 in March 2017. The system was in operation in all 17 hospitals at the end of 2017.

Observed damages

a. **Time**: No damage. The system was deployed 3 years after contract, much on schedule.
b. **Investment cost**: No damage. The estimated cost was 2,800 M DKK including internal costs for education of end users and customization. The SW cost was around 1,100 M DKK. This was much as estimated.
   Operating costs: Operating EPIC for the 17 hospitals costs around 200 M DKK annually. It was expected that costs would be lower than operating the 30 existing systems, but it turned out to be almost the same. When choosing EPIC, the high operating cost was considered balanced with the better user interface. As explained in B2 below, this is dubious.

c. **Business results**: The plan was that the system should give a net benefit per year of around 600 M DKK two years after deployment, but it wasn't specified how. At the time of writing (April 2018) it was too early to see any effect. Even November 2019, no benefit had been reported (see below). One goal was that clinicians could save 1-2 hours per day by not having to log in and out of many systems. This goal is met, although management haven't noticed. However, learning the new system and making custom extensions for all the specialties were much harder than anticipated. Doctors couldn't serve as many patients as earlier. Reporting to the government systems was faulty because doctors now had to do it, and they didn't know how to. Results from lab systems were often lost because doctors didn't know how to order them. This required much more time than the 1-2 hours saved by logging in on many systems. The consequence was additional costs for the hospitals because some departments had to employ more doctors. Waiting lists grew. Young doctors worked overtime, but didn't record it, fearing it would harm their career.
   Quality improvements: Fourteen quality factors were expected to improve, for instance: Better care flow for patients, more successful treatments, and better patient safety. This has not been proved yet. Initially there were rumors that patient safety had decreased, but this was not the case.
d. **User satisfaction**: Job satisfaction was expected to increase, but this has not materialized at present. Initially, job satisfaction became low. The situation is slowly improving.
e. **Feasibility doubts**: No damage.

State at the time of writing (April 2018)
The system is in operation in the 17 hospitals, but doctor's productivity is still not as high as earlier. The benefits were expected to start two years after deployment, which would be mid 2018 in this case. This corresponds to experience with EPIC in other countries. However, today (November 2019) no benefit has been reported.

State November 2019
No benefit has been reported. There is still much dissatisfaction with the system. The planned customization for each specialty has not been made, except for one doctor in one hospital. He customized the system for his own use. His result is very convenient and efficient (see reference 3 below). However, his version hasn't spread to the rest of the department.

References
1. Metcalf-Rinaldo, O. and Mosko Jensen, S.: Learnings from the implementation of Epic, 77 pages (July 2017), http://www.itu.dk/people/slauesen/
2. Region Hovedstaden, September 2013: ITX rapport for vurdering ved anskaffelse af Sundhedsplatform (in Danish).
3. Søren Lauesen, December 2017: Valg af EPIC (in Danish): **http://www.itu.dk/people/slauesen/SorenDamages.html**

Analysis

Cause A1. Doesn't identify user needs and win-win

Analysts knew a lot about current operation and potential benefits, but didn't look carefully at the varying demands of the different medical specialties and local practices.

They didn't create a win-win situation for the clinicians. As an example, doctors find it annoying to record diagnosis codes and other structured data, rather than a verbal explanation. At the same time they want to be able to make research on the effects of treatments. It might have been possible to motivate them by showing examples of research data that would be available if they had structured data.

Effects on damages: Business results, User satisfaction.

Cause A2. Requirements don’t cover customer needs

The requirements specification was around 900 pages with 1800 requirements plus 400 pages with use cases. The requirements looked much like other traditional requirements with long lists of features the system must have. The use cases were just an elaborated version of the same but specified also details of data to be seen or recorded. The problem is that we cannot see the work context in which each feature is used. The result is often that although the system has these features, it is very cumbersome to use.

However, requirements played a small role in the EPIC project because EPIC was an existing system (COTS - commercial off the shelf). The important thing was the criteria used for selecting the supplier. One criterion was looking at the system in real work contexts and assessing how well it supported the work. This was done carefully with each of the three proposals (see the introduction to the EPIC case above and F3 below). However, the evaluation covered only safety aspects, e.g. how sure the doctor is that he sees the right patient on the screen. It didn’t cover performance issues, such as time spent on a consultation.

Effects on damages: None.

Cause A3. Describes the solution in detail. No freedom to the supplier

The many requirements can be seen as describing the solution in detail. Large parts of the requirements described the IT specialist's visions about service oriented architecture, formal descriptions of data, etc. The IT specialists got none of this, because the winner was a COTS system with a confidential data model, etc. However, it didn't cause damages in this case because the winning proposal was selected by other criteria (see cause A2 and F3).

Effects on damages: None.

Cause A7. Wants everything at once

The customer wanted to start with two hospitals 7 km apart, but recently merged into one organizational unit. It was called a "pilot test", but actually 8,000 clinicians were involved. Further the customer wanted to cover all medical specialties and for each specialty define standard procedures across all the hospitals. At the same time they

wanted to change work practice so that doctors didn't dictate anymore, but recorded notes themselves. They also had to order lab-tests themselves and record data for settling accounts. Earlier the secretaries had done it. So the plan was to change many things in several areas at the same time.

Standard procedures across hospitals had been debated in the medical community for years, and there were at least two conflicting schools. The customer found the EPIC project a good opportunity for reconciling these conflicts. They set down a task force of 300 clinicians (around 15 for each medical specialty) who had to come up with data (contents) for the clinical work. As departments had to release clinicians for this, they selected clinicians who were less important. The supplier commented that the competences of these 300 clinicians were dubious and that the work lasted too long.

Although the plan was that all clinicians should stop dictating, it ended up with 400+ doctors at the national hospital (Rigshospitalet) being allowed to continue dictation. This makes sense since it is the hospital with the most complex and varied treatments. It also shows that dictation is possible with the system, and much dissatisfaction could have been avoided by allowing others to do it too. Transition could then come later, building on experiences from other departments.

Effects on damages: Business results, User satisfaction.

Cause A8. Doesn't plan the new work processes

The new work was planned, but not in detail. In particular it wasn't analyzed how the clinician's work would be in the future, how quality would change and how much time a consultation would take initially and later.

Effects on damages: Business results, User satisfaction.

Cause A10. Surprising rule complexity

Danish health authorities pay hospitals according to diagnoses and patient contacts. However, the rules are very complex and changing. Earlier the secretaries knew the rules and recorded everything. Now the doctors had to do it. They had not got the necessary practice and made many errors.

The health authorities also operate systems that the health record systems have to integrate with. One example is a central record of each citizen's medications (FMK). When the doctor starts a patient visit, the system must retrieve the patient's medication record, and when he closes the session, the system must update the central record. The technical protocol was cumbersome and slow. Changes were needed, but it took a long time to implement them.

Effects on damages: Business results, User satisfaction.

Acquisition

Cause B2: Wrong selection criteria

The supplier was selected based on what lawyers call the “economically best proposal”. This is an excellent approach if done correctly, but in this case (and most other cases the author has seen) it is done in a way that doesn’t reflect economy.

Each system was used in a trial hospital department by three teams of doctors, nurses and secretaries. The team got one day of training in the new system, followed by two days of practice. EPIC was clearly better than the competitors (score 7.9 vs. 5.9 and 4.8). The scores were given on questions such as “it is easy to get overview of the patient’s data” and “it is clear which patient the data relates to”.

The systems were also evaluated by 450 clinicians who saw the suppliers' sales demonstrations. EPIC was number two here, but the differences were small (the best got 7.83, EPIC 7.15).

In addition, scores were given for many other aspects such as clinical functionality and IT architecture, each of them assessed by a separate committee. All scores were given on a scale from 0 to 10. Also the cost of the system was transformed into a score between 0 and 10. All of these scores were then weighted and added to a total number of scores. EPIC got the highest total and was selected.

The problem is that these scores don’t reflect the economic value of the proposal. In particular, the areas where the benefit was expected to materialize had no score. As an example, there were no points for how fast it was to order medicine or type notes, or how much time would be saved by not logging in and out of many systems. The most disputed new way of working was that doctors had to write notes for the record themselves, rather than dictating to the secretary. During trial operation at a hospital, this was tested in only one case, and coincidentally by a doctor who always had loved to write long notes himself.

The correct way is to assess the benefit in $ of each proposal and subtract the total cost of the system. The result is the net benefit. Next, choose the proposal with the highest net benefit – or the highest net benefit per $.

The way it was done, the winner could be a system that was an economic loss to the organization. However, we don’t know whether EPIC would have been the winner if a proper cost-benefit analysis had been made.

Effects on damages: Business results, User satisfaction.

Design

Cause C1. Doesn’t ensure usability

New screens or setups were developed for many medical specialties. Although clinicians participated, the result was in many cases cumbersome screens. No usability tests were made.

Programming

Cause D2. Surprises with system integration

Integration with lab systems, with FMK (the government's mandatory database of who got which medicine), etc. caused many problems. In total EPIC had to integrate with 20 other systems plus an undefined number of medico-technical systems. The supplier had given a fixed price per system for this, but ended up spending much more

time than expected. The problems included finding out about the data and the protocol, but also getting the necessary permissions sufficiently fast.

The customer actually had specialists in these areas, but didn't offer help to the supplier.

Effects on damages: Initial business results, Initial user satisfaction.

Test

Cause E1. Deploys the system with insufficient testing

Testing of integrations was incomplete. As an example, the technical communication with FMK (the national medication record) turned out to be very slow and it took a long time to improve since the FMK organization had to be involved. As another example, the customizations for each specialty were not tested for medical correctness and usability, particularly for error handling.

Effects on damages: Business results, User satisfaction.

Deployment

Cause F1. Deploys the system with insufficient support or training

The customer insisted on a big bang for the two, united hospitals. The supplier agreed, although he would have preferred a single hospital. For administrative reasons, it would be too cumbersome to deploy the system in one or a few departments only.

The customer had created many change management groups 9-12 months before deployment. They had followed the development and the planned courses. However, they didn't know what the new work procedures would be, and the user interfaces were not ready, so it could be abstract talk only.

Since rescheduling hadn't been done when development and customization was delayed, deployment became very compressed. As an example, super users couldn't help other users when the system went live, because they had been trained on an earlier, incomplete version. Many clinicians reported that they had to use the system without any training or support.

Effects on damages: Business results, User satisfaction.

Cause F2. Doesn't check whether the system is used as intended

Nobody checked whether the doctor's recordings were correct and whether they used the system in an efficient manner. In principle, super users could have done this, but they didn't have the knowledge as explained for F1. Even some months after deployment, many users would be happy to have an expert watch what they do and help them improve.

Effects on damages: Business results, User satisfaction.

Cause F3. Wrong estimate of human performance

Many doctors spent vastly more time on typing and clicking than on the previous dictation to the secretary, who then updated the record. This decreased performance and user satisfaction. Some doctors even resigned from their job. A small-scale pilot operation could have revealed how big the problem was.

In principle, the problem could have been detected before the contract was signed. Each of the three proposals were tested in a trial department, and it should have been possible to detect the low performance at that point in time. However, according to the test report (ref. 2 above), only ease-of-learning was assessed, not task efficiency. The test version didn't include integration with other systems (would have been very costly to test). Twelve different scenarios were tested, but in just one of them the doctor wrote complex notes (a psychiatry case). Furthermore this doctor happened to love writing notes himself and had always done so. Medicine was ordered in only four of the scenarios, and in two of these it is obscure whether the doctor or the secretary did it. The doctor didn’t settle accounting in any of the cases. These issues later turned out to make the work cumbersome.

Effects on damages: Business results, User satisfaction.

Management

Cause G1. No business goals - or forgets them on the way

It had been known for many years that clinicians spent 1-2 hours per day to log in and out of several systems to see lab results, etc. It would be an obvious goal to save this time, since EPIC integrated with these systems. However, the goal had been hidden in the larger goal of task efficiency, so nobody checked on it. Fortunately, the goal was met. This makes it even more surprising that doctors reported that they couldn't handle as many patients as earlier.

In general, the original goal of task efficiency had a low priority during acquisition and deployment (see F3). Other factors dominated.

Adjusting the business case during acquisition, might have changed priorities for deployment, for instance about customization and standardization, and whether to allow dictating to the record.

Effects on damages: Business results.

Cause G2. Doesn't reschedule, but assumes the rest can be compressed

Early activities had taken longer than expected, e.g. integration and custom extensions for the specialties. However, under heavy pressure from management and supplier, the system was deployed on time and cost. One argument was that if we don’t deploy on time, we will loose the benefit for that period. However, it was not clear what the expected benefit was. Instead of a benefit, the result was dissatisfaction, low user performance and low quality (causes E1, F1 and F2).

Effects on damages: Business results, User satisfaction.

Cause G6. Cashes the benefit before it is harvested
It was planned that 20% of the doctor's secretaries would be dismissed since doctors would take over their work. In many hospitals they did it before the system was deployed, only to realize that doctor's had troubles doing the secretaries work. In November 2017, many departments brought back the secretaries to improve performance.

Effects on damages: Business results, User satisfaction.

Cause G9. Too large steering committees/working groups without competencies
Users with little IT-knowledge customized the user interface. Much effort was spent on reaching consensus. Further the result wasn't tested with other clinicians for medical correctness and usability. See also cause A7, C1, E1 and F1.

Effects on damages: Business results, User satisfaction.

Cause G10. Excessive user involvement
Much time was spent on keeping users informed about the new system and getting their comments - before there was something real to present. Although some consultants say it is important to keep users informed, it can easily become a waste of time.

Effects on damages: No serious damage was observed in this case.

Method for data collection
(Not available)

6. Cures for each cause

Analysis of the 5 projects has revealed 37 different causes of damage. An early analysis of 5 other government IT projects (Bonnerup/ Teknologirådet, 2001) revealed 13 causes. They correspond to the following 13 causes in the present analysis:

A2, A3, A4, A7, A8, B1, C5, F1 (partly), G1 (partly), G4, G7, G8 (partly) and G10.

This section looks at the causes one by one and suggests possible solutions, i.e. ways to prevent the problem. The causes are ordered according to the development activity where they tend to occur. Causes relating to Analysis are identified with A1, A2 …, causes relating to Acquisition are identified with B1, B2 …

Analysis

Cause A1. Doesn't identify user needs and win-win

The cure is to study what exists today and plan the future work processes (cure CA2). Describe the findings as problem-oriented requirements (cure CA1).

Cause A2. Requirements don’t cover customer needs

The cure is to use problem-oriented requirements, cure CA1.

Cause A3. Describes the solution in detail. No freedom to the supplier

Traditional requirements specify the solution - what the system shall do. The supplier has no freedom to do it his way. As a result, the supplier cannot use his existing system unless he adds a lot of functionality. Often the customer's solution turns out to be bad, and then he has to pay for changing it. Problem-oriented requirements avoid these issues. They don’t specify what the system shall do, but what it will be used for (cure CA1).

Cause A4. Makes heavy demands and believes it is for free

The customer is rarely aware what things cost, and may unknowingly ask for very expensive solutions. Requirements with *open target* specify what the customer roughly wants (e.g. around 99.8% availability), but leaves it to the supplier to specify what he can offer (e.g. 99.5% at 5 M DKK per year, alternatively 99.9% at 15 M per year). Open target requirements are part of SL-07 (cure CA1).

Another reason for the heavy demands may be many stakeholders or expert groups, each of which comes up with all the wishes they can dream of. The result is a requirement spec of 1000+ pages divided in 20+ documents that don’t match. The Health record spec is an example. IT-architects wrote their view of the world, security experts theirs, quality experts theirs, surgeons theirs, etc.

Another cure is *scope management*, where the customer keeps track of how many *function points* he asks for. Function points are a way of measuring the size of the project in a technology-independent way (cure CA3).

Cause A5. Oversells technology, e.g. SOA, web-based, workflow engine
New technology is risky for a large project. See cure CA5 (Check technology) and CG7 (ask expert developers).

Cause A6. Multi-vendor strategy - avoid monopoly
With old IT systems, the government and the municipalities had experienced that the supplier essentially had a monopoly. The customer's new systems couldn't get access to existing data and only the original supplier could modify or extend the system. The customer couldn't even transfer his own data to a new system.

Using several suppliers seemed to be a way out. Each supplier delivered part of the big system. The customer imagined that if he wasn't satisfied with one of the suppliers, he could buy the part somewhere else and "plug it in". This was never done in practice. Instead the customer experienced that now he was fighting several monopolies instead of one.

We have two cures here: One about keeping the ownership of data, cure CA6 (Central database & flex SOA). The other about avoiding monopoly by ensuring that third-party can add to your new system and that you later can move your data to another system, cure CA7 (Third-party integrate & exit).

Cause A7. Wants everything at once, e.g. cover the entire country or all types of debt
Deploying a large or complex system is always risky. Many things can go wrong, even when we have been very careful with planning and testing. All the projects in this study had severe problems at deployment. One cure is to reduce the risk by deploying the system for a small number of users (a pilot test) or without using the results (a deployment test), see CE1.

If the system has complex functionality, e.g. many rules, the suggestion is to deploy it part-by-part, initially doing the rare, but complex cases manually (CF1).

Cause A8. Doesn't plan the new work processes
If we don't plan the new work processes and user situations, it is hard to create a system that efficiently supports the new processes. We won't find out until the system is deployed - and then we can do little about it. Early planning of the future work processes helps (cure CA2). Using problem-oriented requirements (CA1) will force analysts to find out early and describe the new processes and user tasks.

Cause A9. No feasible solution. Data missing, performance dubious, etc.
Sometimes the customer imagines solutions that aren't feasible. For instance it may not be possible to serve the necessary number of users, the necessary data has low quality or doesn't exist, or no supplier can deliver what we ask for.

The cures are to carry out an early proof of concept (POC, cure CA8) and involve domain experts and expert developers in the right way (see cure CA4 and CG7).

Cause A10. Surprising rule complexity
For some systems the customer doesn't know the complex rules, but he can find a supplier who does. This is for instance the case with payroll systems. In the

requirements, the customer can just ask the supplier to handle the union salary agreements. The customer doesn't have to turn them into traditional requirements.

For other systems the customer should know the rules, but cannot specify them as requirements. There are two ways out:

1. Automate the simple cases. Let the expert users handle the remaining cases. This was done successfully in the Land Registry project. It might have been done in the EFI case too (debt collection), but was not attempted (see also CF1, deploy part-by-part).
2. Carefully analyze the rules and specify them with some of the many techniques available in software engineering. See cure CA4.

Acquisition

Cause B1. Supplier too optimistic - you must lie to win

The cure is an early proof of concept (POC). See cure CA8.

Cause B2. Wrong selection criteria

Traditional requirements are long lists of functions the system must have. The supplier replies by ticking off the functions his system has. Usually he ticks off everything, because his system has something like the desired function (the customer won't find out until much later). The customer just accepts this and chooses the winner according to price. *We don't have the time to look at the systems*, he says. He doesn't realize that requirements can be met in good and bad ways. Not surprisingly, he later finds out that the system he got, isn't what he expected.

The advice is to use problem-oriented requirements (cure CA1) and let the supplier show how his solution meets each requirement. It takes a few days to assess each solution in this way, but it ensures that the customer knows what he gets.

Often the supplier is selected based on score points rather than money. The health record system is an example. The problem is that the scores don’t reflect the economic value of the proposal. Often the areas where the benefit is expected to materialize get no score at all.

The correct way is to assess the benefit in $ of each proposal and subtract the total cost of the system. The result is the net benefit. Next, choose the proposal with the highest net benefit (or the highest net benefit per dollar). If the net benefit is negative, reconsider whether you need a new system.

Problem-oriented requirements (cure CA1) gives examples of how to make the assessment and how to make sure requirements suffice for harvesting the benefit.

Cause B3. Wrong cost estimates

Wrong cost estimates are usually related to bad requirements. If the requirements don't allow the parties to assess the number and complexity of user screens, data classes and integrations, you cannot estimate the number of function points. And then it is pure guesswork to estimate the price.

The advice is to measure function points for the tailor-made parts of the system (cure CA3). You can measure function points directly from SL-07 requirements (cure CA1).

Some costs are simply forgotten. In the travel card project, the supplier forgot or ignored the cost of back-office systems. In the PolSag project, the customer forgot to include operational costs. A simple checklist for items to include in the business case could have prevented this (see cure CB1).

Design

Cause C1. Doesn't ensure usability

Usability (ease-of-use) is mostly ignored. Some analysts specify low-level requirements to the user interface, e.g. the Back key must always work on web pages, drop-down lists must be used where relevant. Unfortunately, a user interface may meet all these requirements, yet be completely un-intuitive.

Another approach is to develop the system and at the end have a graphical designer make it look nice and colorful. However, this doesn't make the system easy to use. Sometimes the project team makes a few usability tests at the end of development, but at that point in time only minor things can be changed. As an example, a cumbersome user interface needs redesign, and it cannot be done at the end. See cure CC1 for a better and cheaper way to ensure usability. Sometimes developers know the method, but don't use it.

Cause C2. Designs user screens too late

Usability specialists agree that it is important to make early prototypes (mockups) of the user screens, make usability tests of them, improve them and test again until the result is satisfactory. See cure CC1.

Cause C3. Accepts the solution description without understanding it

After the first part of development, the supplier traditionally delivers a solution description that the customer has to sign off. However, there is no agreement among IT-specialists about the contents of a solution description. Often the supplier delivers a bunch of technical specifications, which the customer doesn't understand. Since the customer doesn't know what to expect, he signs off.

The advice is to require early prototypes (cure CC1). The solution description should contain user screens (mockups are okay) and documentation of usability tests. The customer can relate to this.

Cause C4. Cannot see how far the supplier is

This issue is related to C3. The customer doesn't know what to look for when assessing progress. Early prototypes make him able to see progress (cure CC1). The delivered function points are also good indicators (cure CA3). Looking at the suppliers' records of work hours can give a warning if too few hours are spent, but it doesn't help when many work hours are spent, but in a wrong way. Monitoring the remaining work hours can help here (cure CC2), so can talking to expert developers (CG7)

Cause C5. My way without considering the supplier's way

The customer may have a solution in mind and insist on this being delivered. This can make the solution unnecessary expensive. The advice is to use problem-oriented requirements (CA1) and compare them with the customer's idea and the supplier's proposal. If the customer's idea isn't significantly better, use the supplier's proposal. This issue is related to cause A3 (describing the solution in detail).

Programming

Cause D1. Supplier accepts the expensive requirements interpretation

Sometimes requirements are ambiguous and can be met in the cheap way the supplier expected, or the customer's way, which later turns out to be very costly. This occurred in the Travel Card project and in PolSag. In both cases the supplier accepted the costly solution and lost millions. The advice is to always assess the consequences of such choices in terms of cost and delivery time (see cure CG1, replan).

Problem-oriented requirements (CA1) vastly reduce ambiguity of requirements, because the parties can see what the solution is to be used for.

Cause D2. Surprises with system integration

System integration is always a high-risk area that can be very costly to deal with late in the project. The advice is to make a POC (cure CA8). This means trying to exchange data with the external systems. Sometimes it can take months just to get the necessary permissions to connect to the external system.

Another issue is how to test the integration. Does the external system provide data that may be used for testing? And how can your system write data to the foreign system for testing? You will need permissions for this too.

The customer (and his consultant) should take responsibility for these permissions and get them before acquisition. All of this is not easy. Cures are needed.

There may still be some surprises at the end, particularly how to handle errors in the data exchange. Catch them with a deployment test or pilot test (cure CE1).

Test

The test process itself can be a problem, for instance if there is no platform to test on or no test system to integrate with. As an example, if you are developing software for trains, you will need a train and a track. However, we haven’t seen testing as a cause of damage in these 5 projects.

Cause E1. Deploys the system with insufficient testing

Deploying a large system is always risky. How complete is the technical testing? Talk to expert developers to hear their report on test coverage, stress test, etc. (cure CG7).

Many other things can go wrong, even when we have been very careful with planning and testing. The advice is to run a pilot test (or deployment test) with real users, real data and real environments (see cure CE1).

Deployment

Cause F1. Deploys the system with insufficient support or training

When a system is deployed, a lot of problems turn up and the support organization has to sort them out. In many cases support becomes overloaded. The advice is to minimize the problems in these ways:

1. Plan the new work in detail (see CA2).
2. Make early usability tests for eliminating problems and estimating how many problems will be left for support to handle (see CC1).
3. Make a pilot test or deployment test (see CE1).
4. Deploy part-by-part (see CF1).

Cause F2. The system is not used as intended

Deployment implies that users will work in another way than today. Often they don't find out on their own and use the system in a cumbersome way, which ruins user satisfaction and business goals. See cure CE1 (pilot test) and CF2 (follow-up study).

Cause F3. Wrong estimate of human performance

Often users turn out to be unable to work as fast as assumed in the plan. This caused damages in the Land Registry as well as the Health Record. The solution is to test the human performance in a POC (cure CA8) and a pilot test/deployment test (cure CE1).

Management

Cause G1. No business goals - or forgets them on the way

Development takes so much attention that managers often forget about the business goals. The result is that the benefits don't materialize at the end. See cure CG3.

Cause G2. Doesn't reschedule, but assumes the rest can be compressed

When surprises turn up during the project, the time schedule will usually change. Managers tend to assume that they can just compress the rest of the project and still meet the deadline. In the health record case (EPIC), deployment was done on time and cost after heavy pressure from management and supplier. The result was low productivity and angry users. See cure CG1.

Cause G3. The project grows without anybody noticing

This happened in PolSag and in many other projects. The problem is that management doesn't have good measures of progress and of what remains. See cure CA3 and CC2.

Cause G4. Doesn't face the danger. The risk assessment downplays the danger

In some of the projects, the correct risks were identified, but downplayed. The result was damages. See cure CG4.

Cause G5. The financial incentive disappears and the parties fight instead of cooperate

Prevent it by observing whether the project grows (see cure CA3 and CC2). If it does, re-plan before fighting starts (CG1), then look for solutions - or close the project. SL-07 section K2 shows various actions you can take. SL-07 comes with a problem-oriented contract. It shows what you can do if you want to close the project. It has an appendix 2 with a payment plan that ensures there is money left to do the rest.

In government, closing a project is a disaster. In industry it is common and sound business.

Cause G6. Cashes the benefit before it is harvested, e.g. sacks employees/experts too early

This is related to G3 (project growing) and cure CG1 (re-plan) may help. But other means are needed too. Once employees know they are likely to be dismissed, it is hard to keep the best. We have no cure to suggest for this damage cause.

Cause G7. Lack of management involvement

Some managers believe they can manage an IT-project without knowing about development and technicalities. Can a commander-in-chief manage a battle without knowing what goes on in a battlefield? Hardly. IT projects are similar. Without IT project knowledge, the manager cannot be involved. He has only two choices: report delays to his own boss and ask for more money - or suggest that the project is closed and the battle lost.

The advice is to give management the necessary IT insight. See cure CG6. SL-07 comes with a problem-oriented contract. According to its section 3.4, the supplier may ask for a replacement of the manager.

Cause G8. Excessive management or expert involvement

Amazingly, a project can also suffer from too much management involvement. The manager should realize that there are areas where he hasn't the necessary expertise or facts. User interfaces are a good example. Managers sometimes decide that user screens must be in his way - in contradiction with what usability tests show.

In some cases management leaves user involvement to an expert user, who then decides what the requirements are. This often fails because the expert user has little understanding of other stakeholders' needs. We have no cure to suggest for this damage cause. The SL-07 contract’s section 3.4 allows the supplier to ask for a replacement of the manager or expert user, but it will hardly work.

Cause G9. Too large steering committees/working groups without competencies

The solution is to make a small task force and give it the necessary authority. This was the way the Travel Card finally delivered something. See cure CG2.

Cause G10. Excessive user involvement

Users must be involved, yes, but in the right way. See cure CG2.

Cause G11. Believes law blocks sound approaches, e.g. talking to suppliers or running pilot tests

There are many rumors about what is allowed and what is not. In the Land Registry project, management used as an excuse that it was against the law to start with a pilot test, rather than deploying in the entire country. The result was a chaotic deployment. Even if it was against the law, they could have asked for an exemption.

In the debt collection case (EFI), the project owner (Tax) got a warning from the State Lawyer, that withholding salary in the way they did, was dubious. Instead of getting clarification or exemption, they closed the most important way of collecting debt.

See cure CG5 (find constructive lawyers and make them part of the team). We believe something more is needed too, but have no suggestion.

Cause G12. Insufficient staffing

Insufficient staffing can be lack of heads and/or lack of qualifications. In the five projects and other projects the author has studied, we haven't seen lack of heads. In two cases the author has seen lack of experienced people. One was in the Land Registry, where the two experienced people planned to lead the project, were bought by another company. This caused an initial project delay of half a year. The other was a pension project, where the supplier had to deliver a new, cheap system that replaced his old costly system. He had a financial benefit of delaying delivery.

Part of the cure is to monitor remaining work (CC2). The other is to ensure that the contract allows the customer to terminate the contract if staffing is insufficient or internal deadlines violated (SL-07 contract section 2.3 and 3.4).

Cause G13. Doesn't find the root cause

When the project is in big trouble and management doesn't understand why, the consequence may be that the project is closed for being *too complex* or *too ambitious*.

A famous case is the Denver International Airport baggage system (1994) [13]. The system was crucial for fast transfer of passengers and their baggage from one flight to another. The system never performed and delayed opening the airport for years. Management closed the project because it was "too complex and too ambitious". However, external consultants had tested an isolated loop of the total conveyor system. This too didn't work, indicating a simple technical root cause, but nobody reacted.

Lack of finding the root causes was part of the reason for closing the police case-management system. It was also the cause of several years of delay in the Travel Card project (2007 to 2010).

Cure: Ask expert developers (CG7) and give management IT insight (CG6). And don't allow *complexity* or *ambition* to be damage causes. Find the root cause.

7. Cures

Below is a description of each suggested cure. They are ordered according to the development activities where they are primarily used. Cures relating to Analysis are identified with CA1, CA2 …, cures relating to Acquisition are identified with CB1, CB2.

Analysis

Cure CA1. Problem-oriented requirements (SL-07)

Traditional requirements (IEEE 830) are long lists of functions the system must provide. But we cannot see who will use the system, when and for what. Nor can we see what problems users and other stakeholders have today and expect the system to remedy. As a result the customer can get a system that meets all requirements, but doesn't cover his needs. The system is awfully cumbersome to use, doesn't remedy the present problems, and doesn't meet business goals.

Problem-oriented requirements don't describe what the system shall do, but what it will be used for. User stories claim to do the same, but cover only a small part of the requirements. It is also rather arbitrary how "big" a user story is. Does it cover a simple interaction with the system, e.g. seeing the patient's diagnoses? Or getting an overview of the patient' situation, e.g. diagnoses as well as lab results, medication, and other treatments? Or supporting a full consultation in the clinic? Or supporting the entire flow of the patient treatment? And which of these are "epics" rather than user stories?

With SL-07 requirements there is no doubt: It uses *task descriptions* rather than user stories or use cases. A task description must cover everything the user can do from trigger (e.g. start of consultation), without essential interruptions until end (e.g. we cannot do more for the patient right now). So the correct task is *perform a consultation.* The system must support this task, including subtasks such as seeing the patient's diagnoses and ordering medicine.

Kuhail & Lauesen [2] have experimentally shown that task descriptions are much stronger than use cases and much better cover the customer's problems.

Task descriptions are typically 30% of an SL-07 requirements specification. SL-07 also tracks business goals to solution visions and specific requirements, such as critical tasks to support well. SL-07 also specifies data needs, system integration, security, response time, usability, how to select the best supplier, etc. All the requirements are problem-oriented, i.e. the customer writes his needs, and the supplier writes the solution he proposes.

From version 7, SL-07 includes a contract and requirements to the acquisition process (also problem-oriented). Contract as well as the full requirements are available for free. You may copy and modify them to suit your own project.

The author's web site http://www.itu.dk/people/slauesen/SorenReqs.html
has several examples of SL-07 in practice. The textbook *Lauesen: Problem-oriented requirements SL-07 – Guide and contract* [6] is built on a real-life health record

system. The *Y-Foundation* project [5] is another example: The requirements specify a web-site for applicants, combined with a system to support case management, meetings in the assessment board, payment and integration with banks and the tax authority's system. The requirements, the supplier's proposal, supplier selection, issues and disputes during the project, are available.

Open target requirements. The customer is rarely aware what things cost, and may unknowingly ask for very expensive solutions. Requirements with *open target* specify what the customer roughly wants (e.g. around 99.8% availability), but leaves it to the supplier to specify what he can offer (e.g. 99.5% at 5 M DKK per year, alternatively 99.9% at 15 M per year). Open target requirements are part of SL-07. They are another way to write problem-oriented requirements.

Cure CA2. Study as-is, plan to-be

The cure is to do what the books say about the analysis work, but do it carefully: Study what real users do and make sure to study all the stakeholders, e.g. for a health record system all the medical specialties as well as nurses, secretaries and various kinds of patients.

Also plan what users will do in the future with the new system. Try to make all stakeholders feel they get a benefit with the new system (win-win). This is a good way to overcome resistance to change.

However, the weak point in the books is how to describe the findings. The findings must end up as requirements, otherwise they will not be taken into account during development or selection of a new system. Piles of use cases and user stories are too vague to serve as requirements. The recommendation is to describe the findings as problem-oriented requirements (see cure CA1).

Cure CA3. Scope management and function points

Function points are a way of measuring the size of an IT project in a technology-independent way. As an example, a medium complex user screen is 10 function points, a medium complex data class is also 10 function points. In Denmark a function point costs 2,000 to 4,000 $ if the system is developed from scratch. Counting this way, the customer can keep track of the cost of his wishes.

With *scope management* you keep track of the function points while defining the requirements. Whenever you come up with new requirements, you ask yourself how many function points are involved and whether the benefits match the cost. You can also use scope management during development to keep track of how many of the function points have been delivered and how many remain.

Problem-oriented requirements specifications (SL-07) make it easy to estimate the number of function points (cure CA1). With SL-07 you must have a logical data model, and thus the number of data classes. You must also describe the user tasks. Theygive a good indication of how many user screens you need, and how complex they are.

Cure CA4. Domain expert involvement

Domain experts and expert users often know more about the existing work than the ordinary users and the project team. They can point out the existing problems, the complexities and the many special cases the system must deal with. Often they also have good ideas about what to do in the future.

It is important to involve them early on, for instance to ensure that a proof-of-concept deals with the real complexities. But don't let them run the project (see damage G8).

IT experts with experience from similar systems can also be important, for instance to balance the supplier's sales talks or the consultant's favorite solutions.

Systems with complex rules need a combination of domain experts and IT experts. In cooperation they should analyze the rules and specify them with some of the many techniques available in software engineering, for instance state-transition diagrams, tables of combinations, mathematical formulas. Expertise and time is needed to do this. This was done successfully in the travel card project, but only partly in the debt collection project.

Cure CA5. Check technology

The IT industry creates hype all the time and tries to sell it to consultants, who sell it to customers. The hype often originates from researchers who have tried the idea in small scale, where it may seem promising. Sometimes it scales up, but often it turns out to create disasters when used in full scale or without understanding how to use it.

One example is Service Oriented Architecture (SOA), which postulated that you could freely define a landscape of systems and connect them by services, and that you could define these services up front. It turned out that this was very costly and error prone. See more in CA7.

Another example is workflow engines, where you for instance can define how a loan application moves from person to person and which data each person needs. It turned out that there were so many exceptions in real workflows that they were hard to define and automate.

For the customer it is hard to guard against hype, particularly when his consultant advocates the hype. The advice is to seek second opinions and closely study organizations that have solid experience with the new technology.

Cure CA6. Central database + flexible SOA

Splitting a system between two or more suppliers makes you deal with several monopolies rather than one. So the first advice is not to split an application between two suppliers unless there is a very good reason, for instance that each system provides advanced functionality that the other system doesn't have.

The second advice is that the customer takes ownership of data and maintains it in a central database. Third-party suppliers can integrate with this database using services. The third-party can access the central database on demand, or maintain a copy of the central data. He decides.

This approach has been used successfully for 10+ years by STAR (Styrelsen for Arbejdsmarked og Rekruttering). Their central database integrates with around 50 third-party systems. In fact they use the old Database Management approach from the 80'ies. Their approach is that each third-party negotiates specific services with the data owner. Any change in the service has to be negotiated again. In other systems, e.g. the Debt Collection (Tax), there was no central database and a third party had to negotiate with many other parties. The cost was 100,000 - 300,000 DKK per service.

The third advice is to define flexible services rather than specific ones. It allows third-party to define a query without having to negotiate with the data owner. This can be done with Odata services (Open data) that allow (almost) any SQL-query to be made against the data owner's data. We know of only one supplier in Denmark that uses this approach (Scanjour, now part of KMD). They report that it dramatically reduces integration costs and often speeds up data transmission with a factor of 30.

Cure CA7. Third-party can integrate + exit strategy

When you acquire a new system, you will soon want to integrate it with another system from a third party. The supplier of your new system has to be involved. He has a monopoly and can charge you accordingly, unless you have contracted on something else. Your requirements (part of the contract) should specify this:

a. Third parties must be able to integrate with the new system (with the customer's permission).
b. An exit strategy that allows the customer to transfer his own data at any time. In this way the customer can buy another system, convert the old data and feed it into the system.

It is amazing that very few requirements specify this. The consequences are serious, but don't show up until many years later. SL-07 gives detailed examples of such requirements.

Cure CA8. Early proof-of-concept (POC)

A proof of concept can be made before signing the contract or early after. The supplier must prove that he can meet the high-risk requirements, for instance adequate response times for 4,000 users, integration with other systems, and adequate usability. The supplier can deliver the proof in many ways, e.g. running his system with 4,000 simulated users or testing usability with a mockup user interface. If he cannot give a convincing proof, the contract will be cancelled.

Honest suppliers welcome this approach. It eliminates the liars.

Should the proof be before or after signing the contract? The customer prefers before contract, but this can be quite expensive for the supplier. Writing a proposal for a large system may cost 1 M DKK. Having to make a POC may cost another million. He has to spend these amounts without knowing whether he gets the contract. This scares many good suppliers. Making the POC after the contract, means that the supplier is sure to get the contract on the condition that he meets the POC. Further, it allows a close cooperation between the two parties.

The SL07 contract shows details of how to specify and use a POC.

Acquisition

Cure CB1. Cost checklist

Guidelines and templates for writing business cases often contain checklists for what to include as cost elements, e.g. product cost, product operation and maintenance, internal costs for analysis and acquisition, internal costs for testing, internal costs for user training, costs for related hard- and software. Use them and improve them if they don't suffice.

Design

Cure CC1. Early prototypes, usability test and iteration

Usability specialists agree that it is important to make early prototypes (mockups) of the user screens, make usability tests of them, improve them and test again until the result is satisfactory. Usually two or three iterations suffice. Research shows that any programming made at this stage, will make it hard to improve the user interface because it seems too costly to throw away programs. Research also shows that if you follow the specialist advice, total development will be faster and cheaper. You spend a few weeks early, but save months at the end. See Vinter & Lauesen [12].

If analysts have written problem-oriented requirements, they have described the user's future tasks and the data the system must store (cause A2). Based on this, it is possible to design mockup screens and test them for usability. See Lauesen [3].

Cure CC2. Monitor remaining work hours

One way to measure progress is to monitor remaining work hours. Developers typically report per week how much time they have spent on various activities. Make them also report how much time they believe is left on each of the activities they work on. At some point in time, the sum of time spent and time remaining starts to increase. This is a sign of danger. Find out what is wrong.

Programming

No new cures seem necessary.

Test

Cure CE1. Pilot operation / Deployment test

A large system has many users and/or a lot of functionality. Deploying it is risky. Many things can go wrong, even when we have been very careful with planning and testing. All the projects in this study had severe problems at deployment or pilot test. We can address the problems in two ways:

1. Make a *deployment test* that looks like real production, but the results are not used.
2. Or make a *pilot operation* with real work being done, but only a small number of users. Observe how much support is needed and scale up for full operation. We can better overcome helping a small number of users and then improve the process for the next many users. This is not for free, because we somehow have to keep the old and new system running at the same time. The cost of this should be matched with the risk-reduction we get.

The difference between deployment test and pilot operation is whether true work is done. During a deployment test, the work is done in the old way, but also in the new way. The results of the "new way" are not used, however. The deployment test has low risk, but is more costly.

Experienced developers warn that the tests must include the complex cases. All too often only the simple cases are tested. As a result there are severe problems when complex cases occur in real operation. Cure CF1 gives suggestions.

Deployment

Cure CF1. Deploy part-by-part

Deploying a complex system is risky. It is hard to plan for all the complexities. We can address the problems in three ways:

1. Deploy only part of the system, for instance a part that gives an immediate and visible effect. When the situation has settled, deploy more. In the health record case, the first step could be to save 1-2 hours per day by not logging in and out of many systems. Later we can try letting some doctors record data rather than dictate, etc.
2. Automate the simple cases. Let the expert users handle the remaining cases. This was done successfully in the Land Registry project. It might have been done in the EFI case too (tax collection), but was not attempted.
3. Deploy the system for only a small user group. We can better recover from errors that occur a few times and better support few users. This is similar to a pilot test (see cure CE1).

Cure CF2. Follow-up study

A follow-up study is done after a short period of normal use. It investigates whether the business goals are met – and why not. Often missing business goals are caused by users not using the system as intended. The cure is to let expert users observe what the users do, and then find ways to make them do it right.

Management

Cure CG1. Re-plan

When surprises turn up during the project, the time schedule will usually change. Managers tend to assume that they can just compress the rest of the project and still meet the deadline. The result is usually a chaotic deployment. The advice is to re-plan whenever a surprise turns up. Re-planning involves available resources, dependencies, cost, schedule and risks. Re-planning may also involve re-scoping the project, e.g. less functionality or fewer user groups.

Re-planning is hard to do in government projects due to political pressure. In industry re-planning is common.

Cure CG2. Small task force with authority

Managing a large IT project needs more than one person, although only one of them will be the responsible manager. The reason is that many expertizes are needed. But a large task force has little decision power.

A task force (project team) of 3 to 5 persons is ideal. Together they should cover the business aspects, all user aspects, the IT aspects and preferably the legal aspects. The task force should listen to all kinds of users and other stakeholders, but should not let them decide.

Users must be involved, yes, but in the right way. Having lots of users debate and trying to agree, rarely succeeds. But listening to their needs and complaints is important. Using them as test subjects in usability tests and deployment tests is mandatory.

Cure CG3. Monitor business case

Development takes so much attention that managers often forget about the business goals. The result may be that the benefits don't materialize at the end.

The solution is to monitor the business case regularly during development. As an example, once you have selected the supplier, the business case may change. Adjust it and look for new opportunities. In general, review the business goals whenever you review the risks.

Cure CG4. Correct risk management

Identify the important risks, e.g. by means of check lists and experiences from similar projects. For each risk find ways to observe whether the risk materializes. As an example, if the risk is project delay, find ways to measure whether progress is ok. Next plan what to do in case it happens – compressing the rest is rarely a solution. Don’t downplay the risk by saying it is unlikely. Plan what to do – just in case.

Cure CG5. Constructive lawyers

Analysts and developers are generally scared of lawyers. Whenever they ask a lawyer, the answer is usually that what they want to do isn't allowed. To the lawyer, this is the safe answer, but it can block progress. The advice is to find constructive lawyers (they exist) and make them part of the project team. The team has joint responsibility for coming up with a solution.

Cure CG6. Give management IT insight

In order to involve managers in a project, they need to understand what goes on below the surface. Managers don’t have to learn programming, but they should understand what a computer system does and how it interacts with the organization. Further they should be able to detect when problems like the ones in this report turn up, and know how to prevent them and what to do about them if they turn up. We welcome suggestions for courses that give managers this insight.

Cure CG7. Ask expert developers

In many cases managers try to resolve problems on their own level, e.g. replacing staff or asking for schedule extensions. This may be ok if they know the root causes, but often they don’t. In order to find the root causes, they should ask the supplier’s expert developers, and if possible, their own. These talks may need patience from both sides.

8. References

References that apply to a specific project are shown after the introduction to that project. The most important ones are shown below too.

1. Bonnerup/The Technology Advisory Board (March 2001): Experiences from government IT projects - how to do it better (in Danish).
2. Kuhail, M. and Lauesen, S. (2011): Task descriptions versus use cases. Requirements Engineering Journal, DOI 10.1007/s00766-011-0140-1.
3. Lauesen, S. (2005): User Interface Design - A Software Engineering Perspective. Addison-Wesley, 2005.
4. Lauesen, S. (2012): Why the electronic land registry failed. Proceedings of REFSQ 12, Springer Verlag. Also available at: http://www.itu.dk/people/slauesen/
5. Lauesen, S. (2018a): Problem-Oriented Requirements in Practice - a Case Study. In: E. Kamsties et al. (Eds.): REFSQ 2018, LNCS 10753, pp. 3–19, 2018, Springer, https://doi.org/10.1007/978-3-319-77243-1_1.
6. Lauesen, S. (2018b): Problem-oriented requirements SL-07 – Guide and contract v7, 2019, ISBN: 9781791748906. Also available at: http://www.itu.dk/people/slauesen/SorenReqs.html.
7. Metcalf-Rinaldo, O. and Mosko Jensen, S.: Learnings from the implementation of Epic, 77 pages (July 2017), http://www.itu.dk/people/slauesen/
8. Rigsrevisionen (National Auditors): Beretning til Statsrevisorerne om det digitale tinglysningsprojekt (Report to the State Auditors on the Electronic Land Registry project), 74 pages (in Danish) (August 2010).
9. Rigsrevisionen (National Auditors): Beretning til Statsrevisorerne om rejsekortprojektet (Report to the State Auditors on the Travel Card project), 47 pages (in Danish) (June 2011).
10. Rigsrevisionen (National Auditors): Beretning til Statsrevisorerne om politiets it-system POLSAG (Report to the State Auditors on the police IT system POLSAG), 56 pages (in Danish) (March 2013).
11. Rigsrevisionen (National Auditors): Beretning til Statsrevisorerne om SKATs systemmodernisering (Report to the State Auditors on Tax's system renovation), 34 pages (in Danish) (January 2015).
12. Vinter, O. and Lauesen, S.: Preventing Requirement Defects: An Experiment in Process Improvement. Requirements Engineering Journal (2001) 6, pp. 37-50. Springer Verlag, 2001.
13. Montealegre, R. and Keil, M.: De-escalating information technology projects: Lessons from the Denver International Airport: MIS Quarterly, Vol. 24, No. 3 (Sep 2000), pp. 417-447.