DamageCaseStories2.docx
Søren Lauesen 07-05-2017

Draft

Damage and damage causes in large IT government projects

Søren Lauesen
slauesen@itu.dk

1. Land Registry 4
2. Travel card, RK 12
3. Police case management, PolSag....18
4. Debt collection, EFI....23
5. Health record system, EPIC....29
6. Possible cures 35

This report is a first draft. It is intended for discussion and idea generation in a larger audience. There may be errors in various places. Please send me a mail if you encounter errors or have ideas for curing the damage causes.

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 candidates 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.

The report identifies 36 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: Teknologirådet/-Bonnerup (2001): Experiences from government IT projects - how to do it better.

The report also shows the type of failure (something the team didn't know, knew but ignored, was misinformed about, etc.). It also gives suggestions for what could have been done, for instance writing requirements in a different way, running pilot tests, monitoring the business aspects, etc.

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 has a proven solution to the customer's problems - also to the problems he 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.

The report consists of:

1. An Excel file with three spreadsheets: *Type-cure* with a row for each of the 36 damage causes. Columns for projects; who failed; damage caused; failure type and suggested cure (if known). *Cures* with a column for each potential cure and indications of which causes it would cure. *VisibleDamages1, 2* . . . with several columns for each project, one for each visible damage with indication of which cause contributed to this damage. The spreadsheets will print nicely on A3 portrait sheets.
2. This paper which gives the story behind each project and how the damage causes happened. It also lists all the 36 causes and what can be done about them.

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.

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 performance: The business case became worse than planned. This can be less profit. It can also be other business factors, such as increased treatment time for patients 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 - 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.

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 entered online or through system-to-system interfaces to financial IT systems. It was estimated that 30% of the registrations would be selected for manual checking.

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 performance**: The first 7 months after deployment, registry took 50 days for 30% of the registrations. The rules said 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 only for the Land Registry, 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: See a full description of the case in:

Lauesen (2012): Why the electronic land registry failed. In Proceedings of REFSQ 12, Springer Verlag. Also available at: http://www.itu.dk/people/slauesen/

Rigsrevisionen (National Auditors), August 2010: Beretning til Statsrevisorerne om det digitale tinglysningsprojekt, 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 performance, User satisfaction.

The registry project also included registration of car loans based on a new external 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, this implementation 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. Doesn't write requirements that 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 only for inspiration. 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 costs and development time to grow.

Effects on damages: Time, Cost, Business performance, 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 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 costs 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 think about 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 are 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 this, but the customer insisted because citizens had to be treated equally. 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 took 50 days instead of 10.

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 need for authorizations and the effect of usability issues.

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 performance, User satisfaction.

Acquisition

Cause B1. 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 performance, User satisfaction.

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 only understandable to a domain expert. And this turned out to be the case also in this project. 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 performance, 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 performance, User satisfaction.

Cause C5. 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. 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 the customer to see how far the supplier is. What should he ask for? Hours spent or technical descriptions? Doesn'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 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
This was obviously an important cause of damage in the Land Registry.

Effects on damages: Business performance, 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 non-IT citizens.

Effects on damages: Business performance.

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 performance.

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 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 not made a system for public use. With internal systems, they could easily support the users, but a system for large-scale public use was very different. The risk analysis just downplays the danger.
Risk 3: As explained in cause C5, 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 used correctly. There were no safe-guards and nobody took action for the high risks.

Effects on damages: Time, Cost, Business performance, 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 performance.

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 legal 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 had to be.

Effects on damages: Business performance, User satisfaction.

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 performance**: 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:

Rigsrevisionen (National Auditors), June 2011: Beretning til Statsrevisorerne om rejsekortprojektet, 47 pages (in Danish).

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 give 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. Doesn't write requirements that 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 only at the SW issues, 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 fail 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 B2. 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 B5. Forgets costly components
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, but didn't know what to expect. His reaction was: EW has promised to deliver, so we wait and see. So 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 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 C5. 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 performance.

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 had 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. Money runs out 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, only with RK. 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.

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 performance**: 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 police (RP) still don't know what to do about case management.

References:

Rigsrevisionen (National Auditors), March 2013: Beretning til Statsrevisorerne om politiets it-system POLSAG, 56 pages (in Danish).

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. 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 the old supplier's staff had often 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 performance, Supplier loss, Feasibility doubts.

Cause A2. Doesn't write requirements that 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.

Later, the requirements became "design-level requirements" that specified the technical solution in detail, e.g. what each button on the screens would do. 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 performance.

Cause A3. Describes the solution in detail. No freedom to the supplier
As explained in cause A2, the requirements were 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 for 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 performance.

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 performance.

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 again. 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 that showed an incredible number of database calls. Everybody could see that this was 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 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 B4. Wrong cost estimates
Scanjour had estimated their price based on the 100 existing screens that had to be redesigned in the Scanjour way. They couldn't see how much special functionality was behind each screen. Also the high costs on SOA were a surprise.

Effects on damages: Time, Supplier loss.

Cause B5. Forgets costly components
The customer forgot to include the cost of operating the new system during development (68 M DKK), 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: Cost, Business performance.

Design

Cause C6. 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.

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 reported 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.

Deployment

Cause F1. Deploys the system with insufficient support
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.

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 got no answer.

Further, nobody had paid attention to the growing cost of maintenance (see cause B5). As a result, nobody could come up with a reason to continue the project.

Effects on damages: Business performance.

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. Money runs out and the 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 performance, Feasibility doubts.

Cause G9. Too large steering committees/working groups without competencies
The 40 policemen participating in the design didn't work out.

Effects on damages: Time, Cost, Business performance.

Cause G10. Excessive user involvement
As cause G9.

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.

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 amounts of debts anyway. Initially (2005) all bailiffs and other specialists were organizationally moved to the tax department and around 1000 employees dismissed.

The idea was that a new IT system (EFI) would automate the debt collection and spare further jobs (estimated 300 jobs). 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 couldn't report debts correctly to EFI.

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 and the project closed after two more years.
b. **Cost**: The estimated project cost was 144 M DKK. The actual cost became around 600 M DKK when the first part of the system was deployed.
c. **Business performance**: The plan was that the system should save salaries for additional 300 bailiffs and speed up debt collection. Debt had increased to around 70,000 M DKK. The bailiffs were actually dismissed, saving around 120 M DKK per year, but collection didn't succeed and soon stopped completely.
d. **Feasibility doubts**: When the project was closed, the feasibility of the IT architecture was questioned. Accenture wrote a report that clearly said that the programs involved couldn't be rescued.

State today

The system was gradually deployed from September 2013 and closed down September 2015. Debts were not collected or collected against the law. At the time of writing (March 2017), Tax tries to start all over, focusing on only the most important types of debt.

References:
Skatteministeriet (Ministry of Tax), September 2015: Redegørelse om ét fælles inddrivelsessystem, 58 pages (in Danish).
Rigsrevisionen (National Auditors), January 2015: Beretning til Statsrevisorerne om SKATs systemmodernisering, 34 pages (in Danish).
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*. Actually, bailiff's had many tricks and concerns in their duty. 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 do 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.

Another issue was that existing data on debts had poor quality. Documents were missing and dates incorrect. As long as debt collection was manual, this wasn't a problem. But when it is automated, it becomes a big problem.

Effects on damages: Time, Cost, Business performance.

Cause A2. Doesn't write requirements that cover customer needs
The early contract with the supplier was rather open. It essentially said: *Help us find out what to do*. In the later contracts with fixed prices, requirements were defined 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 performance.

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 and EFI didn't even work. 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.

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 get access to 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 transactions with Skat (DMI) and one that collected the claims. 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. This only worked in simple cases.

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.

Instead of starting with a few debt types that could collect most of the money, 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 performance.

Cause A9. No feasible solution
During analysis, developers had only looked at simple flows for debt collection. Was it possible at all to specify the entire logic, e.g. when debt was outdated and the debtor paid it anyway, or claims changed as a result of court cases? And was it possible and realistic to transfer debts from old manual debt collection files to EFI?

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.

Effects on damages: Time, Cost, Business performance.

Cause A10. Surprising rule complexity
To map some of the complexity, 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 400 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. In principle this spreadsheet could have been interpreted by a program that carried out the rules as specified. This would have put much of the burden of testing on the tax and debt experts.

However, this wasn't done, maybe because rules interfered with each other in a debt-type dependent way. Instead each debt type was implemented by itself.

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

Acquisition

Cause B4. 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 between EFI and DMI.

Effects on damages: Time, Cost, Business performance.

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. It was time to show some results. 9,700 test cases were planned, 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 280,000 cells in the spreadsheet. Developers tried to explain, but management wouldn't listen and later claimed that they hadn't been informed.

There was no pilot test that could check whether full deployment was acceptable.

Effects on damages: Business performance.

Deployment

Cause F2. Doesn't check whether the system is used as intended
EFI received debts to collect from external users, such as municipalities and transport operators. Tax insisted that it was these users' own responsibility to select the proper debt type (out of the 400 possible) and supply the necessary documentation. The system didn't check anything.

One result was that bailiffs often met with debtors, but were unable to document the debts.

Effects on damages: Business performance.

Management

Cause G1. No business goals - or forgets them on the way
- The business goal for EFI was weak: *Efficient debt collection*.
- 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 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 performance.

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.

It seems obvious that management didn't understand the real problems behind the continuous requests for added funding. Much was blamed on the suppliers, and they probably overcharged significantly. 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 collection specialists was needed. When deployment was decided, management ignored the warnings about missing tests and high risks.

Effects on damages: Time, Cost, Business performance, 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 performance.

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 performance.

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 clinician had to find paper folders and log in and out of several systems. 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 used in a trial hospital department by doctors and nurses over a period of three weeks. They were also evaluated by 450 clinicians who saw the suppliers' sales demonstration. Based on this and to some extent also the cost of the system, 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 data 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 March 2017. All 17 hospitals are expected to have the system at the end of 2017.

Observed damages

a. **Time**: No damage. The system was deployed 3 years after contract, much on schedule.
b. **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 planned.
c. **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. Although this is a bit dubious, we will not consider it a damage at this point in time.
d. **Business performance**: The plan was that the system should give a net benefit per year of around 600 M DKK two years after deployment. At the time of writing (March 2017) it is too early to see any effect. 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 had to work harder and couldn't serve as many patients as earlier. Reporting to the government systems was faulty because doctors had to do it now, 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 of overtime payment, etc.

e. **Quality improvements**: Fourteen quality factors were expected to improve, for instance: Better care flow for patients, more successful treatments, better patient safety. This has not been proved yet. Initially there were rumors that patient safety had decreased, but this was not the case.
f. **User satisfaction**: Job satisfaction was expected to increase, but this has not materialized at present. Initially, job satisfaction became low.

State today (March 2017)
The system is in operation in 8 hospitals, but it will take some time to get the doctor's productivity as high as earlier and have accounting sorted out with the government. The benefits were expected to start two years after deployment, so it is too early see any. This corresponds to experience with EPIC in other countries.

References:
Oliver Metcalf-Rinaldo, Stephan Mosko Jensen, January 2017: Learnings from the implementation of Epic, 77 pages (Master's thesis).

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 performance, Quality improvement, User satisfaction.

Cause A2. Doesn't write requirements that 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 the we cannot see the work context in which each feature is used. The result is often that although the system has this feature, 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. The primary 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. The system was used in a trial hospital department by doctors and nurses over a period of three weeks. There was no doubt to clinicians that EPIC was the most efficient to use.

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. However, it didn't cause damages in this case because the winning proposal was selected by other criteria (see cause A2).

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 they 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 performance, Quality improvement, 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 it would take initially and later.

Effects on damages: Business performance, Quality improvement, User satisfaction.

Cause A10. Surprising rule complexity
Danish health authorities pay hospitals according to patient treatments. 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 make them.

Effects on damages: Business performance, User satisfaction.

Acquisition

No damage observations.

Design

No damage observations.

Programming

Cause D2. Surprises with system integration
Integration with lab systems, 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 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 performance, 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 performance, Quality improvement, User satisfaction.

Deployment

Cause F1. Deploys the system with insufficient support
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 only one or a few departments.

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 only be abstract talk.

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 performance, Quality improvement, 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 performance, Quality improvement, 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. The project team has not yet been able to explain why.

Effects on damages: Business performance, 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. 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 performance.

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. The result was a compressed deployment that caused E1, F1 and F2.

Effects on damages: Business performance, Quality improvement, 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, E1 and F1.

Effects on damages: Business performance, Quality improvement, 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.

6. Possible cures

Analysis of the 5 projects has revealed 36 different causes of damage. An early analysis of 5 other government IT projects (Teknologirådet/Bonnerup, 2001) revealed 13 causes. They correspond to the following 13 causes in the present analysis:

A2, A3, A4, A7, A8, B2, C6, F1 (partly), G1 (partly), G4, G7, G8 (partly) and G10.

This section looks at the causes one by one and outlines possible solutions, i.e. ways to prevent the problem.

Analysis

Cause A1. Doesn't identify user needs and win-win
The cure is to do what the books say about the analysis work, but do it carefully: Study real users 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).

However, the weak point in the existing methods 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 the 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 cause A2).

Cause A2. Doesn't write requirements that cover customer needs
Traditional requirements 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.

The need to use "less detailed requirements" has been mentioned by others, but they have not given examples of what such requirements could be.

The cure is to use **problem-oriented requirements**. They describe what user and system should achieve together and which problems to remedy. They also track business goals to solution ideas and specific requirements. In his proposal, the supplier specifies how his system supports the users and remedies the problems. Several examples of how to do this in practice exist and have been used successfully. *Lauesen: Guide to requirements SL-07* is one such example, combined with explanations of why it is done this way. It covers all kinds of requirements, including system integration, security, response time, business goals, etc.

Cause A3. Describes the solution in detail. No freedom to the supplier
Traditional requirements can be so detailed that they specify a solution. 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 (see A2).

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 (see A2).

Another solution 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. 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 15-20,000 DKK, if the system is developed from scratch. Counting this way, the customer can keep track of the cost of his wishes.

Cause A5. Oversells technology, e.g. SOA, web-based, workflow engine
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. 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 other organizations that have solid experience with the new technology.

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 three advices here:
1. The first advice is not to split an application between two suppliers unless there is a good reason, for instance that each system provides advanced functionality that the other system doesn't have.

2. 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. This approach has been used successfully for 10+ years by STAR (Styrelsen for Arbejdsmarked og Rekruttering). In fact it is the old Database Management approach from the 80'ies.

3. The third advice is to specify these two things in the requirements for a new system:
   a. Third parties must be able to integrate with the new system (with the customer's permission). In this way the new system can be extended with radical new functionality without having to negotiate with the old supplier.
   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.

Cause A7. Wants everything at once, e.g. cover the entire country or all types of debt

Deploying a large 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 or pilot test. We can address the problems with two means:

1. Deploy the system for only a small group of users or run a pilot test (see F1). 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.
2. 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 (EPIC), 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 having some doctors record data rather than dictate, etc.

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 (see A1). Using problem-oriented requirements (see A2) will force analysts to find out early and describe the new process. If we have ignored this, we must plan the new work early in the deployment phase.

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 cure is to carry out a proof of concept (POC) where we early in the project set up simulated loads of the potential system, try using the existing data, and ask suppliers what is possible.

We should also involve our existing experts who know more about the complexities and problems than the analysts do. But don't let them run the project (see G8).

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. He doesn't have to turn them into traditional requirements.

For other systems the customer should know, but cannot specify the rules. 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 (tax collection), but was not attempted (see also A7).
2. Carefully 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 EFI project.

Acquisition

Cause B1. Believes law blocks sound approaches, e.g. talking to suppliers or run pilot tests
There are many rumors about what is allowed and what is not. When analysts ask the lawyers, the answer is usually that it 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.

Cause B2. Supplier too optimistic - you must lie to win
The cure is a POC (proof of concept) early after the contract. 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 4000 simulated users or testing usability with a mockup user interface. If he cannot give a convincing proof, the contract can be cancelled.

Honest suppliers welcome this approach. It eliminates the liars.

Why don't we require the potential suppliers to make the proof before sending their proposal? Because it can be quite expensive for them. Writing a proposal for a large system may cost 1 M DKK. Having to make a POC may cost another million. This would scare many good suppliers. Making the POC after contract, allows the supplier to include the cost of the POC in his proposal. Further, it allows a close cooperation between the parties.

Cause B3. The customer doesn't assess the proposals
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 and let the supplier show how he meets them. It takes a few days to make this assessment, but it ensures that the customer knows what he gets.

Cause B4. 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. You can measure function points directly from SL-07 requirements.

Cause B5. Forgets costly components
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. We welcome ideas for how to avoid such errors. Check lists may be an answer.

Design

Cause C1. Dosn't ensure usability, even when they know how
Usability (ease-of-use) is mostly ignored. When developers considers it all, the usual 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 cause C2 for a better way to ensure usability. Sometimes developers know the method, but they 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. 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.

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.

Cause C3. Accepts the solution description without understanding it
After the first part of development, the supplier delivers a solution description, which 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. The solution description should contain user screens (mockups are okay) and documentation of usability tests. The customer can relate to this, and it matches well with the advice in C2.

Cause C4. Replaced by A10

Cause C5. 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. 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 (see G3).

Cause C6. 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 and compare them with the customer's idea and the supplier's solution. If the customer's idea isn't significantly better, use the supplier's solution. This issue is related to 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 that later turns out to be very costly. This occurred in the Travel Card project and 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.

Problem-oriented requirements 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 (proof of concept) right after signing the contract. 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. The customer (and his consultant) should take responsibility for these permissions and get them before acquisition.

There may still be some surprises at the end. Catch them with a deployment test or pilot test (see F1).

Test

Cause E1. Deploys the system with insufficient testing
Deploying a large system is always risky. Many things can go wrong, even when we have been very careful with planning and testing. The advice is to run a deployment test or a pilot test with real users, real data and real environments (see F1).

Deployment

Cause F1. Deploys the system with insufficient support
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 A8).
2. Make early usability tests for eliminating problems and estimating how many problems will be left for support to handle.
3. Make a *deployment test* that looks like real production, but the results are not used.
4. Or make a *pilot test* with real work being done, but only a small number of users. Observe how much support is needed and scale up for full operation.

The difference between deployment test and pilot test 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.

Cause F2. Doesn't check whether the system is 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 the business goals. The solution is to let expert users observe what the users do in a pilot test, or after a short period of normal use (a *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. The solution is to test the human performance in an early POC, trial operation or pilot test (see F1).

Management

Cause G1. No business goals - or forgets them on the way
Development takes so much attention that managers forget about the business goals. The result is often 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.

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. Project managers tend to assume that they can just compress the rest of the project and still meet the deadline. The result is often a chaotic deployment. The advice is to re-plan whenever a surprise turns up. Re-planning is hard to do in government projects due to political pressure. In industry re-planning is common.

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. Here are some advices:

1. Use a scope manager (see A4) to keep track of how much has been developed and how much remains.
2. Monitor remaining work. Developers typically report per week how much time they have spent. Make them also report how much time they believe is left on 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.

Cause G4. Doesn't face the danger. The risk assessment downplays the danger
The advice is simple: Do as the books say. Don't downplay the danger, but find ways to observe whether the risk materializes. And plan what to do in case it happens.

Cause G5. Money runs out and the parties fight instead of cooperate
Prevent it by observing whether the project grows (see G3). Re-plan when fighting starts, then look for financing - or close the project.

Cause G6. Cashes the benefit before it is harvested, e.g. sacks employees/experts too early
This is related to G3 (project growing). But other means are needed too. Once employees know they are likely to be dismissed, it is hard to keep the best. Solutions wanted.

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 can only 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 project competences. We welcome suggestions for such courses. As a minimum, the manager should be able to detect when problems like the ones in this report turn up. And also know what to do about them.

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. Solutions wanted.

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. The task force should listen to all kinds of users and other stakeholders, but don't let them decide (see G10).

Cause G10. Excessive user involvement
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.