How Software Projects Fail

Software appears, by its nature, to be difficult to engineer on a large scale. Nevertheless, there is an insatiable demand for sizeable, well-engineered software.

We continue to be dogged by large numbers of project failures, on small and large projects.

Many of these are due to mistakes in project management.

In this lecture we discuss:

- Examples of project failure on a large scale.
- Instances of mistakes leading to problems on projects I have managed.

Scale of the Problem

Quoted from CIO Magazine Dec 1998:

- Recent Standish Group survey indicates "46% of IT projects were over budget and overdue and 28 % failed altogether".
- "only 24 % of IT projects undertaken by Fortune 500 companies in 1998 will be completed successfully."

A source in the US military claims:

- There is one Army project that is 1 billion dollars over budget, and still has no working system.
- 100% of all DoD projects over 1 million lines of code are delayed.
- One third of all projects are cancelled before completion.
- One half of all projects cost twice as much as they were estimated to cost.
- Documentation of a 350 KLOC DoD project costs about four hundred dollars per page.

License Registration System

In 1990 the Washington State Department of Licensing launched its License Application Mitigation Project (LAMP): $41.8 million over 5 years to automate the state's vehicle registration and license renewal processes.

By 1993 budget was $51 million and system was expected to be obsolete when finished.

In 1997 the plug was pulled, about $40 million having been wasted.

Problems:

- Too big in concept with too few early deliverables.
- Split between in-house and contractor development.
- The organisation didn't want to hear that the project was a failure.

Customer Database System

In 1996 a US consumer group embarked on an 18-month, $1 million project to replace its customer database. The new system was delivered on time but didn't work as promised, handling routine transactions smoothly but tripping over more complex ones.

Within three weeks the database was shut down, transactions were processed by hand and a new team was brought in to rebuild the system.

Problems:

- The design team was over-optimistic in agreeing to requirements.
- Developers became fixated on deadlines, allowing errors to be ignored.

Customer Tracking System

In 1996 a San Francisco bank was poised to roll out an application for tracking customer calls. Reports provided by the new system would be going directly to the president of the bank and board of directors. An initial product demo seemed sluggish, but telephone banking division managers were assured by the designers that all was well.

But the the system crashed constantly, could not support multiple users at once and did not meet the bank's security requirements. After three months the project was killed; resulting in a loss of approximately $200,000 in staff time and consulting fees.

Problems:

- The bank failed to check the quality of its contractors.
- Complicated reporting structure with no clear chain of command.
- Nobody "owned" the software.

Payroll system

The night before the launch of a new payroll system in a major US health-care organization, project managers hit problems. During a sample run, the off-the-shelf package began producing cheques for negative amounts, for sums larger than the top executive's annual take-home pay, *etc*.

Payroll was delivered on time for most employees but the incident damaged the relationship between information systems and the payroll and finance departments, and the programming manager resigned in disgrace.

Problems:

- The new system had not been tested under realistic conditions.
- Differences between old and new systems had not been explained (so 8.0 $ per hour was entered as 800 $ per hour).
- "A lack of clear leadership was a problem from the beginning."

Distribution System

Anticipating growth, a \$100 million division of a \$740 million manufacturing business earmarks \$5 million for a new distribution and customer service system to replace its old one. The project is to take a year and a half to complete. Two years later, the CIO is sacked and a new executive brought in to save the project. Three months later, the system breaks down altogether.

Nine months later, the CIO approached his boss, the CEO to tell him the project is a failure. "It was kind of like telling him a relative had died," he recalls. "First he denied it, then he went through a grieving process, then he accepted it. It was just so much money for a division that size to wave in the wind."

Problems:

- Wrong direction from the start.
- Inadequate software plan.
- Nobody "owned" the software.

Common Project Mistakes

The remainder of this lecture talks about mistakes I have made on projects either producing software or models/specifications of software.

I observe many other people making similar mistakes.

None of the mistakes I describe are directly technical, although the projects in which they occurred were strongly technical and employed formal languages.

Escalating Complexity of Notation

Begin with Horn clauses but

sorts are important so ...

$\Downarrow$

adopt a sorted logic, and by now

we know that composition of

functions is important so ...

$\Downarrow$

extend to a sorted lambda calculus ...

$\Downarrow$

$\vdots$

Each extension has a high cost - not only technical but in terms of re-explaining our methods to domain specialists.

Overpowered Modelling Languages

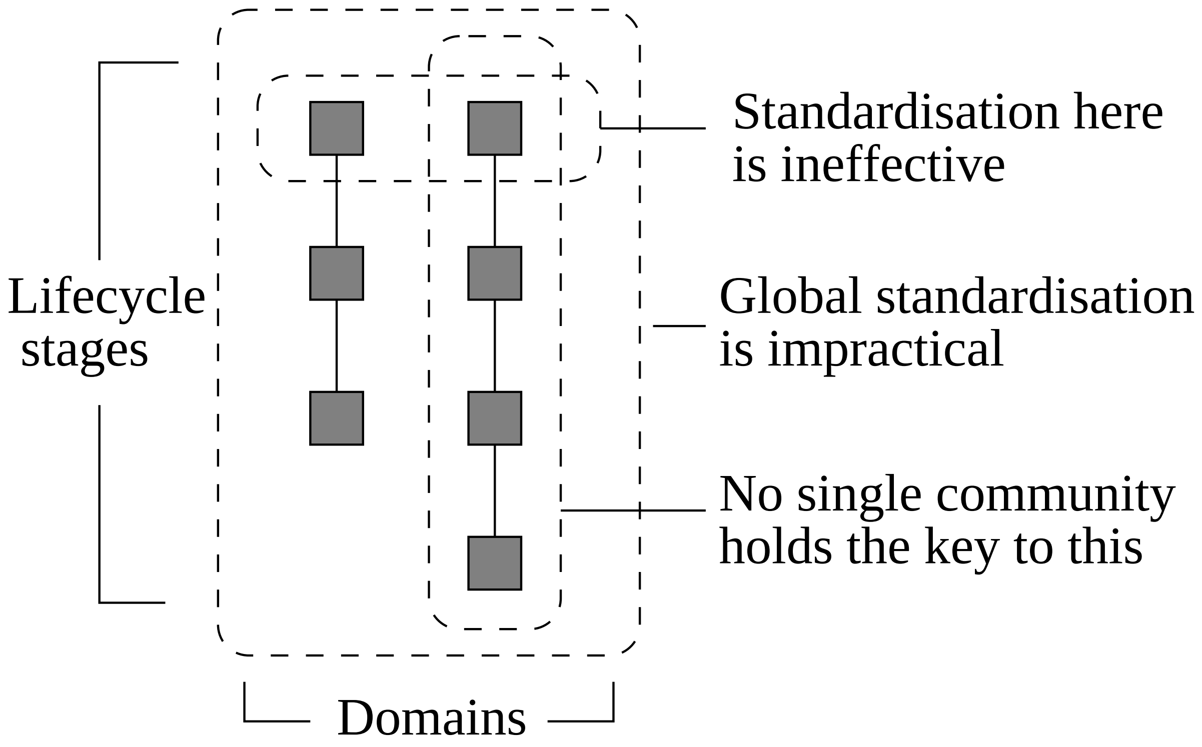
Our early ecological modelling systems synthesised simulation models from ecological problem descriptions.

It is easy to underestimate the effort needed to ensure that any permitted problem description is capable of guiding the synthesiser to a solution.

- If problem description drives generation then cost is in pre-analysis.
- If generation drives problem description then cost is in dialogue control.

Either way, this is (arguably) the main constraint on size of such systems.

Cost of Interfaces



“The User” doesn’t Exist

In some of our early techniques editing work we were sloppy about this. The argument used is:

- Techniques editing deploys structures which are standard in logic programming (true).
- In the right hands, logic programs can be built more quickly and easily using a tool based on these (arguably true).
- So if we build a useful tool there will be a user group for it (false).

Tension Between Breadth and Depth

- In some projects most benefits accrue from a narrowly focused formalisation effort (*e.g.* WWW site generation).
- In other projects formalisation provides a broad framework, connecting informal material (*e.g.* safety system work).

Timing of Benefits Matters

- If we wait too long then the circumstances of our industrial collaborators will change (*e.g.* safety system work)).
- If we deliver too early then we risk over-optimism (*e.g.* early model synthesis work).

Inexperienced Experts

In a project applying temporal logic to industrial process monitoring the following happened:

- We found a niche which clearly suited the theory developed by our temporal logic experts.
- Our industrial collaborators were switched on to the idea.
- We then discovered that our experts had never represented problems with anything other than "A's" and "B's".
- They had to be helped over this hurdle.

Guidance Doesn't Imply Education

When comparing the performance of students using a techniques editor, as opposed to a normal editor, it seems that:

- The techniques group could answer the questions faster.
- They could also produce better code.
- But they weren't any better at problem formulation.

Consensus on Ontology is Rare

Projects which are predicated on "absolute" meaning of symbols and "perfect" knowledge sharing are doomed:

- What does "biomass" mean to biologists ?
- What does $P(X = x)$ mean to those who reason about uncertainty ?

The answers depend on who you ask.

Easy to Know What You Don't Want

Many projects involve prototyping because the technology is new to those in the domain.

An advantage of this is:

- Prototypes focus attention.

But a disadvantage is:

- Prototypes are easy to re-invent in response to criticism, leading to "perpetual prototyping".

The Costs of Evaluation May Be Hidden

Evaluation a structure editor by comparing performance of students using the editor against those using a normal editor. In addition to finding groups of students and examples requires:

- Course material for design using the new editor.
- Shaping of existing course material to avoid unfair comparison.
- People other than the implementors to run the courses.
- Observational studies as well as pre-post testing.
- Testing in advance of full experiment.

Systems Which Can't be Evaluated

Reasons why empirical evaluation may not be possible:

- There are no *bona fide* users.
- Difficulty of interpolation.
- Impossible to delimit the environment.
- Impossible to perform controlled experiments.
- Localised performance measures don't reflect acceptability.

“Experiments” Which Can’t Fail

Questions for which (if you work hard enough) the answer is likely to be “yes”:

- Can a problem tackled in system X be tackled by (similar) system Y ?
- Can system Y be made to work better than X on this problem ?
- Can people be trained to use system X ?
- Will people become better at solving task T using system X ?

We can arrange for the answer to be “yes” because we haven’t given the assumptions under which our systems are designed or used.