Software Projects: Why they Fail and Why There Are Standards

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 despite the fact that we continue to be dogged by project failures, on small and large projects. Many of these are due to mistakes in project management.

In this lecture we discuss some examples of project failure on a large scale, then survey the state of current software-related engineering standards.

0.1 The Scale of the Problem

Horror stories abound of spectacularly unsuccessful projects containing a significant software component. It is difficult to be precise about how much this costs us overall but even the individual examples of large scale failures are worrying. Quoting from CIO Magazine Dec 1998:

- A 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."

This problem does not appear to be localised to the private sector. Public sector projects also go badly wrong. For example, a source in the US military has claimed in the recent past that:

- There was one Army project that was 1 billion dollars over budget, and still had no working system.
- 100% of all DoD projects over 1 million lines of code were delayed.
- One third of all projects were canceled before completion.
- One half of all projects cost twice as much as they were estimated to cost.

Let's take a look at some specific examples of project failure.

0.1.1 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 project was canceled, 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.

0.1.2 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.

0.1.3 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.

0.1.4 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."

0.1.5 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.

0.2 Software Engineering Standards

According to the IEEE Computer Society Software Engineering Standards Committee a standard can be:

- An object or measure of comparison that defines or represents the magnitude of a unit.
- A characterisation that establishes allowable tolerances or constraints for categories of items,
- A degree or level of required excellence or attainment.

Standards relevant to software engineers are motivated by a wide variety of concerns, including these:

Prevents idiosyncrasy : *e.g.* Standards for primitives in programming languages).

Repeatability : *e.g.* Repeating complex inspection processes.

Consensus wisdom : *e.g.* Software metrics.

Cross-specialisation : *e.g.* Software safety standards.

Customer protection : *e.g.* Quality assurance standards.

Professional discipline : *e.g.* V & V standards.

Badging : *e.g.* Capability Maturity Model levels.

Comparatively few software products are forced by law to comply with specific standards. Most software products have comprehensive non-warranty disclaimers. However:

- For particularly sensitive applications (*e.g.* safety critical) your software will have to meet certain standards before purchase.
- US courts have used voluntary standards to establish a supplier's "duty of care".
- Adherence to standards is a strong defence against negligence claims (admissible in court in some parts of the world).
- There are instances of faults in products being traced back to faults in standards, so
- standards writers must themselves be vigilant against malpractice suits.

Figure 1 illustrates the different levels at which software engineering standards may be constructed. The core of standards definitions are element standards, giving the key information appropriate to the standard. These, however,

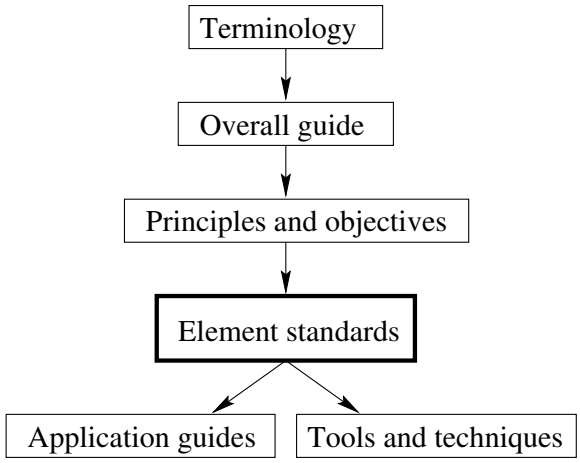

Figure 1: Levels of software engineering standards

normally are a consequence of consensus which required agreement on principles and objectives, terminology and the overall remit of the standard. Supplementing the standard there may also be guides to its application and perhaps to the tools and methods consistent with it.

A variety of organisations have been formed to construct, vet and maintain standards documents. Among the most important of these for software engineers are the following:

ANSI : American National Standards Institute. Does not itself make standards but approves them.

AIAA : American Institute of Aeronautics and Astronautics (*e.g.* AIAA R-013-1992 Recommended Practice for Software Reliability).

EIA : Electronic Industries Association (*e.g.* EIA/IS-632 Systems Engineering)

IEC : International Electrotechnical Commission (*e.g.* IEC 61508 Functional Safety - Safety-Related Systems)

IEEE : Institute of Electrical and Electronics Engineers Computer Society Software Engineering Standards Committee (*e.g.* IEEE Std 1228-1994 Standard for Software Safety Plans)

ISO : International Organisation for Standardisation (*e.g.* ISO/IEC 2382-7:1989 Vocabulary-Part 7: Computer Programming)

If you want to consult an appropriate standard, however, the names of these organisations don't necessarily indicate where to look since different organisations set standards relevant to different aspects of software engineering. Some of the relevant aspects include:

Computer Science Standards (e.g. ISO/IEC 8631:1989 Program Constructs and Conventions for their Representation): Surprisingly few "pure" CS standards exist, although one could argue this is because CS is pervasive in others.

Quality Assurance Standards (e.g. IEEE Std 1061-1992 Standard for Software Quality Metrics Methodology): Differing views of quality standards: taking a systems view (that good management systems yield high quality); and taking an analytical view (that good measurement frameworks yield high quality).

Project Management Standards (e.g. IEE Std 1058.1-1987 Standard for Software Project Management Plans): These are concerned with how general principles of good management are applied to specific areas of software engineering.

Systems Engineering Standards (e.g. ISO/IEC WD 15288 System Life Cycle Processes): Particular application domains develop sophisticated interactions between system and software engineering, so standardising from a systems point of view can be beneficial.

Dependability Standards (e.g. IEC 1025(1990) Fault Tree Analysis): As hardware dependability has improved, software has received more attention as a dependability risk. Dependability of software isn't just a question of internal measures (*e.g.* availability, reliability) but also broader issues (*e.g.* maintainability, system context). Dependability standards often set integrity levels necessary to maintain system risks within acceptable limits.

Safety Standards (e.g. IEC 61508 Functional Safety - Safety-Related Systems): These traditionally come out of specific industrial sectors, since safety requires deep analysis of the domain as well as the technology.

Resources Standards (e.g. IEEE P1320.1 Standard Syntax and Semantics for IDEF0): Although software engineering is in flux, it is possible to standardise on some forms of resources which are used widely across applications.

Product Standards (e.g. ISO/IEC 14598 Software product evaluation): These focus on the products of software engineering, rather than on the processes used to obtain them. Perhaps surprisingly, product standards seem difficult to obtain.

Process Standards (e.g. ISO/IEC 15026 System and Software Integrity Levels): A popular focus of standardisation, partly because product standardisation is elusive and partly because much has been gained by refining process. Much of software engineering is in fact the study of process.

Company Guidelines (e.g. Shell UK Code of Practice: Fire and Gas Detection and Alarm Systems for Offshore Installations): Specific companies may develop their own guidelines for system/software design. These define good practice within a company. They often conform to more general standards.

0.3 Exercise

Look back at each of the project failures summarised in Section 0.1. Would any of the sorts of software related standards we discussed have helped to avoid project failure? If so, how? If not, why not?