Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 1999 Proceedings

Americas Conference on Information Systems (AMCIS)

December 1999

Hard Failures - Soft Solutions

Paul Spedding
University of Salford

Trevor Wood-Harper
University of South Australia

Hard Failures - Soft Solutions

Paul Spedding [1] Trevor Wood-Harper [1, 2]
1: Information Systems Research Centre, University of Salford
2: School of Accounting and Information Systems, University of South Australia

Abstract

The traditional view is that hard problems - those susceptible to the software engineering approach - are in some sense easier to solve than soft problems / problem situations. Hard problems are (in theory) more easily scoped and their solutions more precisely defined and more easily achieved than is the case for soft or messy problems This paper suggests that this is not necessarily the case: reasons for failure in hard and soft environments have much in common. A case study will be used to support the argument

Hard Failure

There is no doubt that the software engineering (SE) approach has achieved a great deal since the term was first used in the late 1960s. The original waterfall model involves a number of clearly defined and separable steps, and as one step is completed the development moves forwards - steps cannot in theory be revisited. Boehm (1988), writing from within the SE paradigm, proposed an improvement, the spiral model, in which earlier steps can be revisited and revised.

No adherent of software engineering would claim however that the approach has prevented the occurrence of failed systems. Indeed the literature has many descriptions of widespread failures, for instance the US Department of Defense survey in 1979 showed (Neumann 1995) that only 2% of software expenditure was on software in satisfactory use, whereas over 75% was on systems that were either delivered and never used or never even delivered.

In such cases, the reaction of both researchers and practitioners has often been to claim either that its methods had not been implemented correctly, or that the approach itself required further improvement.

In the former case, any failure in a system which has been software engineered is viewed as a failure in one or more of its steps. One common reason put forward for the failure of a system is that the customers or users changed their minds about requirements part-way through the development.

In the latter case, many in the SE community believe that updated methods are the key to improved systems. Many improvements have been suggested to the software engineering approach, but none has yet proved to be the illusive "silver bullet" sought but not found by Brooks (1987). Brooks and also Yourdon (1993) agree that in the foreseeable future, no method or tool is likely to appear which will guarantee software quality.

Soft Failure

Adopting a softer approach implies that, for instance, an information system can sometimes be viewed as a failure, even when every aspect of the supporting software system works as its designer intended.

Since information system failure is a complex concept, it is useful to explore IS failure itself before attempting to find the relationship between hard and soft failure.

The ubiquitous failure of information systems is an area of considerable interest to researchers. There is however limited uniformity in the views of researchers over the definition and nature of information systems failure, and also how it can be avoided. For instance, Lyytinen (1988) claims that information systems failure can be viewed as the gap between stakeholder expectation and what is actually delivered: a rich definition owing to the diversity of potential stakeholder perspectives. Lyytinen also suggests that many (arguably) failing information systems are viewed as successful from a harder, software engineering perspective simply because there are no identifiable software design or coding errors.

Robinson (1994) draws on the infamous failure in 1992 of the computerised London Ambulance Service system. He suggests that IS failure is all-too-often viewed as *pathological*, in the sense that the failure could have been avoided by the use of better methods, or by more rational behaviour and greater co-operation between those involved. Robinson instead proposes a social definition of failure, where the differing goals and expectation of stakeholder groups will inevitably lead to differing views on the success or failure of a particular IS outcome.

Orlikowski (1999) draws on the work of Argyris and also Schon and in proposing that the reason why some computer-based information systems fail is in the unbridged gulf between "espoused technology" and "technology-in-use". She suggests that the successful installation of a new system cannot ensure its successful use, and that the ethos of the organisation, for instance the levels of staff co-operation and their view towards

knowledge sharing play the major part in realising the success of the new system.

Most researchers are agreed that information systems failure is subject to many different stakeholder perspectives and is particularly context-rich.

Case Study - the Aerospace Company Research Project

The authors were involved between 1994 and 1997 in a research project exploring software quality assurance (SQA) in a large aerospace company. Many of the details of this project cannot be put into the public domain, but a number of lessons can be learned from the experience gained in the project.

The company was developing a completely new compressed life-cycle model (CLCM) involving new methods and CASE tools for developing airborne real-time software control systems. For instance, the CLCM was to make extensive use of an integrated modelling environment, with widespread usage of sophisticated software tools. The new methods would involve computer-generated Ada code (autocode) being used on safety critical software systems - an innovation for the company, even the industry.

By means of the CLCM, the aerospace company was intending to reduce avionics systems development time by around 40% and cost by 30%, without any reduction in quality. The company saw the development as crucial to the future of the aerospace company and a blueprint for new aircraft projects well into the twenty-first century.

It was seen to be equally crucial for the success of the CLCM that customers (internal and exterior to the company) were themselves "assured" of the quality and safety of the delivered software systems - a particularly sensitive area for real-time aircraft control systems.

Throughout the period, the CLCM was in its research and development phase, and had not been approved for use on real aircraft projects, despite being some years overdue.

Senior staff in the CLCM project were drawn from systems development teams on real aircraft projects and were in most cases highly skilled and professional adherents of the SE approach.

The CLCM project also employed a large number of more junior staff, who, because of the size of the task, often shouldered fairly major areas of responsibility in, for instance, adapting a bought-in autocode generator for CLCM use.

The original research emphasis had been to attempt to apply the methods of traditional hard operational research to SQA within the CLCM project. In what was always accepted by both sides as a challenging (possibly even unrealisable) project, the inputs to the SQA process were to be identified, then measured, as were the quality outputs / benefits associated with the SQA process. The difficulties associated with this emphasis became increasingly clear. It was problematic to measure the inputs (largely staff effort) to the SQA process, and it was even more difficult to measure output in terms of improved quality. In these circumstance, the emphasis increasingly shifted towards modelling the SQA process in the CLCM project - very different from traditional SQA - and a synergy was established here between the work of the authors and that of a team of internal company staff who were modelling processes as part of a company-wide BPR exercise. The methodologies included semi-structured interviews and the participation in process modelling workshops.

In fact, the implementation of CLCM was subject to constant time slips: when the authors' involvement ceased in 1997, the project was around two years behind schedule, and with no realistic hopes of complete implementation within the foreseeable future.

Case Study Research Outcome / Discussion

Although the terms "hard" and "soft" are not precisely defined, their use is sufficiently common to make them clear labels in this discussion.

The outcome of the process modelling was a deepened understanding of the ways in which software quality assurance methods had grown up in tandem with the system development methods they supported, and any change to the latter would necessitate an up-date to the former. For instance, the US Department of Defense demands that airborne software complies with military standard MIL-STD-498, which calls (amongst other things) for code walkthroughs in software modules. This is a well-respected method for software testing, but completely inappropriate for autocode elements, since a walkthrough clearly depends on the contribution of a human programming team.

A further outcome was an understanding of the issues which had caused the company considerable difficulty as it strove to develop and implement its CLCM.

However, progress on the CLCM had been slow since the project had begun in 1993, and towards the end of the authors' involvement in 1997, the project slowed down to the point where no real progress was being made. This remains the situation in 1999 even though the CLCM project is still formally continuing.

Clearly the CLCM would have to be judged as a failure or partial failure.

On the spectrum from soft (exclusively human) to hard (exclusively technical), there is no doubt that any

airborne, safety critical control system is inevitably going to be at the hard end. The CLCM, with its reliance on integrated modelling environment, high-powered CASE tools and (potentially) the auto-generation of safety-critical code is certainly a very hard software-engineered system.

Most previous work in this area has concerned itself with (to a greater or lesser extent) soft developments. However in this case, we have a very hard environment, yet, and this is the central point of this paper, the situation here ties in well with the literature relating to softer environments. In exploring the difficulties identified in the CLCM during the process modelling, it became clear to the authors that the problems were almost exclusively human / organisational. They cannot be explained as either ill-elicited or changing user requirements, nor as poor software engineering methods.

The major sphere of difficulty perceived by the authors was that the aerospace company operates almost exclusively in an environment in which a project (ie the development of a new aircraft) is a very large one-off affair, with many areas of complexity. It consists of many thousands of processes and increasingly it may be a large, inter-organisational, multinational operation, executed within a highly political context. The complexity of such developments is recognised in, for instance Checkland (1981). However, the paradigm adopted by the aerospace company remained almost exclusively that of traditional (software) engineering.

A further concern was morale amongst two groups of staff. This was generally poor, and this undoubtedly affected the potential benefits of the technology.

- staff working on the CLCM project itself were often young and inexperienced; job volatility was high and the project was constantly under threat of restructuring or even cancellation.

- the software QA staff tended to be viewed as "police officers" rather than "consultants": despite the fact that many had considerable quality-related expertise, they were not seen as major contributors to enhancing quality. Instead, the quality assurance process was widely seen as a matter of fulfilling a number of somewhat arbitrary requirements, so that a member of SQA staff would be under pressure to provide a signature to allow the development process to move on to the next stage. There was no doubt that SQA staff were also held in relatively low esteem and were generally not well rewarded.

In fact many staff in the company felt that they had been suffering from an "innovation overload" for a number of years. Following ISO9000 accreditation, and European quality initiatives, the company had also embraced the capability maturity model (CMM) and was a participant in the ongoing SPICE process improvement initiative.

Referring to the literature, Lyytinen or Robinson would identify similar problems in this ultra-hard environment in the distance between the proposed technical solution and the experience of those trying to develop and implement it.

It is perhaps particularly helpful to apply Orlikowski's approach: in this case the **espoused technology** would be the sophisticated CASE tools which were developed in house or bought in, forming a vital input to software developments in the CLCM. A huge technical effort went into attempts to validate and qualify these tools. However, as **technology-in-use**, the system has failed, or is failing, for precisely the "soft" reasons described above.

Conclusion

Even in attempting to solve a hard problem, a traditional software engineering / engineering paradigm carries no guarantee of success. There is concern that the almost exclusive adoption of such a paradigm fails to embrace the organisational and political complexity of a major project such as the development of a new aircraft. It is only by the use of softer approaches that an understanding of "hard" failures is forthcoming.

References

Available on request from the first named author.