Systems Failures: An approach to building a coping strategy

by
A. J. M. Donaldson & J. O. Jenkins

School of Computing Science
Middlesex University, Hendon Campus,
The Burroughs, London, NW4 4BT,United Kingdom
email: John.Donaldson@dial.pipex.com and J.Jenkins@mdx.ac.uk

Presented at the 14th IEEE Conference on Software Engineering Education & Training, February 19-21 2001, Charlotte, NC, USA.

1. Introduction

The phrase *systems failure* strikes fear into some; puzzlement into others; and feelings of "I told you so" into most. When systems fail, they can cause havoc everywhere. They affect the organisations involved in creating, maintaining and using them and they can have a profound effect on the people involved, directly or indirectly. They are highly complicated, they are complex and they may manifest themselves in many different ways. Systems/projects may totally fail, partially fail, or simply "go down" for (e.g.) a working day.

It is important to know that term system failure in the context of this work means: "*any work connected with a system or project (business or otherwise) that is found to have failed, or to be in a position where failure is inevitable, if appropriate action is not taken*". This broad definition is encompasses notions taken from the whole spectrum of systems failures from: low-level software-reliability issues, through to the highest levels of managerial decision-making. This is a philosophical view and is *not* intended to address the individual workings of software code, or specialist knowledge within branches of business management practice.

The causes of systems and project failures, vary considerably. Each case has to be taken in isolation and examined, to see where it has gone wrong in the past, or is starting to go wrong at present. In a true-life scenario, it is essential to be able to predict likely problems that may arise or accurately recognise failure symptoms when they occur. To achieve this it is important to be able to identify what is really going on and when these facts have been established, to be able to select a suitable means of handling the situation.

Two European "Esprit" research projects (ES22292 and ES23066) examined software and multimedia quality practices and provided a framework for addressing these issues. These frameworks did not just promote "best practices" within Software Engineering, but sought to address some of the wider issues of systems and their role within the business. Middlesex University has taken this theme forward with a specific remit to address the subject of Systems Failures.

2. Why is there such a problem, when so much is known about failures?

It is reasonable to ask: why if much is known about what has gone before, is this problem so difficult to overcome? The simple answer is that while many excellent thoughts have been written down (e.g. [1]) describing the failures themselves and a lot has been done to improve systems practice and management, a relatively small amount of research has been specifically addressed systems failures as such. A few exceptions to this may be found in university computing science departments, business schools and government departments in the United States, UK and Europe; but in an area about which so much has been written, this does not represent a lot of "work-in-progress".

Traditional well founded engineering disciplines have developed successful management practice over a very long time. Software Engineering has only evolved during the last half of the twentieth century and its culture is still immature. It is not surprising that attempts to adopt good engineering practices in times of rapidly advancing technology and fierce cost-controlled commercial competition are being met with significant difficulties. The need for rapid development imposes its own constraints on performance, quality, and productivity. Being able to quickly and cost-effectively provide complex software solutions of good quality has become critical in differentiating success from failure.

3. Causes of systems failures

A number of key factors have been observed to contribute towards project failure. Broadly speaking the reasons may be grouped into three categories: requirements engineering, insufficient resources and human intervention; and each of these may be broken down into further sub-categories. The outcome is often a situation where many, disparate, reasons emerge. These may involve incomplete/ambiguous/inconsistent specifications; or poor planning. There may have been no clear assignment of responsibility, lack of adequate tools/techniques, or wrong user-involvement. These complex features interact and while a single factor be the cause, it may the be co-incidence and accumulation of many small, apparently insignificant factors that are to blame.

In addition to this, it has to be remembered that everything is not always what it seems to be. Fashion, expectancy and internal/external politics can strongly influence events. Much publicised catastrophes such as the first attempted Arianne-V launch and the Mars Climate Orbiter loss, showed up problems that were not immediately attributable to a particular system failing. External influences have played a significant part, as in the Denver Airport saga [1], where an already trouble project became inextricably embroiled within the local Political situation. What this all means of course, is that any analysis has to take account of factors which might not at first seem to be remotely connected with the technology in use.

4. Preventing, or rescuing, software system failure: some existing methods.

There are apparent gaps between being able to assemble the facts of what has happened and then moulding them into some kind of formal methodical shape. This is evinced by the fact that so many approaches to solving the problem have occurred through a variety of different views of what "best practice" is and how it may be most effectively developed.

The literature indicates many ways of handling software management that are both quantitative and qualitative. These include existing software quality maintenance and improvement techniques such as GQM [2] and assessment techniques CMM [3] and SPICE [4], while Software Risk Management, [5], takes a pragmatic approach to what is going on.

More recently, a number of other approaches have started to evolve, which address how to assure system success. Three of these are noted as being highly relevant to this work:

– *The Experience Factory* [6], advocates learning from past experiences in order to improve software processes and products. An organisational structure is designed to support and encourage the effective reuse of software experiences.
– *Project De-escalation* [7] is essentially a business-oriented solution which appeals to the hard economics of the situation. Failing systems and runaways often attract additional effort and Project De-escalation addresses this phenomenon with reduced commitment to a failing course of action, allowing troubled projects to be successfully turned around or sensibly abandoned. Examples of its key actions are: changes to top management and real main-board accountability; reducing managerial threats; going public; not punishing whistle blowing.
– *Focussing on the organisation* [8], involves strictly examining the implications of the managerial infrastructure in place and in examining the relationships between the various entities involved and then matches up software engineering practices to that revised structure.

5. Building a systematic approach to solving systems failure problems.

The symptoms of failure can be grouped into areas of commonly encountered causal factors. Typically examples may be: technology; insufficient resources; internal politics; inertia; malicious action; grandiosity; requirements issues; etc. The effects of size are as yet unclear – how the inferences drawn from one type of project, may be used to seek solutions for similar projects on a different scale. Other important factors include the nature the organisation, the technology and software methodologies in use. It is essential to identify the true_causes of why a system/project fails and current work is directed at finding a means of taking symptoms and “distilling” them into the true characteristics of a system problem. In any case, rigorous treatment as opposed to “opinion” is required and this has meant extensive data capture from the literature and field work.

Having assembled the data into a manageable format, there is a whole range of possible ways to assist with understanding these symptoms. The question of selecting a modelling technique was approached with an open mind. At the most formal end of the scale, techniques such Petri Nets [9] were initially considered and discarded on the grounds that the solutions needed also have to be able to address high-level philosophical, often esoteric, arguments. At the softer end of the modelling spectrum, approaches such as the OPEN Method [10] and other ethnographic approaches may be applied to all stages of the development life-cycle. Soft Systems Methodology (SSM) is a qualitative methodology [11] that applies systems concepts to qualitative research by enabling a focus to be made on decision-making, and is well-suited to scenario-based analysis.

In the “centre” of these approaches to modelling techniques is the increasingly popular Unified Software Development Process with its associated UML specification language [12], which is rapidly becoming a standard modelling technique within the field of Software Engineering. Some research has been conducted with a view to extending UML. (a) A version of UML has been created [13] in which task dynamics may be modelled, including temporal ordering of activities. (b) Other work [14] has focused on how UML and SSM may be used in a scenario-based context.

Any solutions in the field of systems failures have to use good sound knowledge to support well reasoned judgements. The availability of facilities such as SWEBOK will suit this need well, but they will only permit access to the best practice; not to any qualitative assessment of the situation at hand. What is needed is a good system of being able to ferret out information and then being able to act upon the knowledge so gained.

The approach being developed uses a mixture of scenarios and fundamental principles of Risk Management [5] as its basis for reasoning. This does not mean searching for similar scenarios based on similar experiences, or creating a scenario based upon the immediate knowledge to hand. It is difficult to predict or propose a general sequence of scenarios as each case's needs vary according to its nature and the constraints involved. To get at the essence of a failure case, a scenario has to be allowed to develop so that the true characteristics of the situation may emerge and so aid a confident resolution

There is not a direct fit between SSM and Use-case modelling because they do not serve exactly the same purpose. Use-case analyses bring out very little of what goes on within a business, but they contain much more operational detail than equivalent SSM models and this can help the analyst to better understand each activity. Through multiple perspectives however, SSM promotes a truly thorough examination of why a business exists and hence helps to more fully identify critical logical decisions. By linking some of these softer approaches to the Unified Software Development Process's principles and paying due regard to the requirements of risk-based reasoning, a comprehensive, robust stance may be adopted. It is then easier for the software practitioner to select a particular course of action in coming terms with the problem.

And finally … this discussion suggests that a broad view has to be taken. There is no reason why alternative approaches cannot be accommodated in the search for a solution and indeed it is blinkered dogma and a lack of pluralist thinking that has led to many disgraceful software catastrophes in the past. Handling failures requires open minds, reception to more than just one type of fashionable approach. To this end it is essential that organs such as SWEBOK are enabled to disseminate this information while at the same time it is incumbent on educators and trainers to provide appropriate wisdom at appropriate levels, for those who are learning in the profession.

References

[1] Glass R.L., Software Runaways: "Lessons learned from massive software project failures", Prentice Hall, ISBN 0-13-673443-X , 1998

[2] van Solingen R. and Berghout E., "The GQM Method: a Practical Guide for Quality Improvement of Software Development", M^{c}Graw Hill, ISBN 007 709553 7, 1999

[3] Paulk M.C., Curtis W., Chrissis M.B., and Weber C. V., "Capability Maturity Model, Version 1.1," IEEE Software, Vol. 10(4), July 1993.

[4] El Emam K., Drouin J-N., and W. Melo (eds.), SPICE: *The Theory and Practice of Software Process Improvement and Capability Determination,* IEEE-CS Press, ISBN 0-8186-7798-8, 1997

[5] Lawrence Pfleeger S., "Risky business: what we have yet to learn about software risk management", Journal of Systems and Software, 53, 2000, pp 265-273.

[6] Basili V. R., "The Experience Factory and Its Relationship to Other Quality Approaches", Advances in Computers, Vol. 41, Academic Press, Inc., 1995.

[7] Keil M. and Robey D., "Turning around troubled software projects", Journal of Management Information Systems, Vol. 15(4), 1999

[8] Doherty, N. F. & King, M., "The consideration of organizational issues during systems development process", Behaviour & Information Technology, 17, 1,1998.

[9] Jensen K., “Coloured Petri Nets: A high-level language for system design and analysis”, in K. Jensen and G. Rozenberg, (eds.) High-Level Petri Nets. Theory and Application, Springer-Verlag, 1991

[10] Graham, I., Henderson-Sellers B and Younessi H., “The OPEN Process Specification”1st Edition, ACM Press, ISBN 0-201-33133-0, 1997

[11] Checkland P., and Scholes J., ”Soft Systems Methodology in Action, Wiley, New York.

[12] Booch G., Rumbaugh J.,Jacobson I., “The Unified Modelling Language User Guide” Addison-Wesley, 1998

[13] Killich S., et al., “Task Modelling for Co-operative Work”, Behaviour & Information Technology, Vol. 18(5), 1999

[14] Bustard D.W., He Z., and Wilkie F.G., “Linking soft systems and use-case modelling through scenarios”, Interacting with Computers, 13, (2000), pp 97–110