# SYSTEMS FAILURES:
# An approach to understanding what can go wrong

John Donaldson & John Jenkins

Middlesex University

John.Donaldson@dial.pipex.com J.Jenkins@mdx.ac.uk

Presented at European Software Day of EuroMicr'00, Maastricht, NL, September 2000.

Published in Proceedings of the European Software Day of EuroMicr'00,, 2000, ISBN 0-7695-0872-4

## *Abstract*

*Software Systems permeate just about every aspect of life throughout developed and industrialised nations today. When failures arise, the aftermath is highly complicated because there are so many ways that they may occur. The complexity of the situation becomes evident even before the apparent or actual causes are considered, especially when the rate of change of the technology is taken into account.*

*Systems failures cause havoc everywhere. They may be total, partial or temporary; and examination of the subject has to consider: implemented systems, systems being enhanced and totally new projects.*

*This paper considers some of the common causes of Systems Failures that are being investigated following the end of Esprit projects (Multispace and Space-UFO). The work takes a more specific look at the quality implications on software systems failures and then goes on to look at current research into what steps may be taken to systematically understand what has actually gone wrong. From that point, possible options open to provide credible and robust solutions to these types of problem, are discussed.*

## *1. Introduction*

Software is used throughout developed and industrialised nations today. It is at the root of applications ranging from sophisticated military control systems and real-time applications in surgery through multimedia and business systems to the chip technology that is now involved in familiar utensils of life such as cell-phones, photo-copiers, and a host of recreational devices.

When failures arise, they are highly complicated because there are so many ways that they may occur. The complexity is evident even before the apparent or actual causes are considered. For example: there is the question of whether or not it is the system itself that is in trouble or a project that is being done to create or enhance a system. There are questions of whether the system: totally fails as with the notorious London Stock Exchange Taurus project [1]; partially fails as with one current UK. Government project [2], or simply “goes down”, as with the London Stock Exchange where a temporary software failure caused the trading system to be unavailable for seven hours on 5th April 2000. As this 3-dimensional picture builds up, its complicated nature becomes even greater, when the rate of change of the technology is taken into account. The problem is recognised as being a serious one [3] and many attempts have been made at rectifying it.

The Esprit projects Multispace (ESPRIT project 23066) and Space-UFO (ESPRIT project 22292) examined software quality practices and provided a framework for addressing these issues. This work has been taken forward with the specific aim of looking into the causes and nature of systems failures. Systems failures cause havoc everywhere. In the USA, the Federal Aviation Authority (FAA) has attempted without success over fifteen years, to install an "all States" Air Traffic Control system [4]. A similar protracted situation has arisen in the UK with Air Traffic Control [5] and is one of the reasons given by H.M. Government for proposing the partial denationalisation of Air Traffic Control within the UK. One particularly interesting outcome from the initial work has been the way that the perception of what happens in Europe is different from what happens in the United States and the fact that where there is a cross culture between Europe and the USA then quite different problems may to emerge[6].

Failure causes 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 predic3t 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.

## *2. Causes of systems failures*

A number of key factors have been observed to contribute towards project failure [7], [8]. Broadly speaking the reasons may be grouped into three categories: requirements engineering, insufficient resources and human intervention.

Each of these may be broken down into further sub-categories and sub-sub- categories although it should be recognised that as the examination of the specific cause becomes more detailed, then overlaps become apparent across the overall three categories. These three may in turn be broken down into different constituent elements, some of which may be apply in each of the three categories. The outcome is a situation where many, often disparate, reasons emerge. These may involve incomplete, ambiguous, and inconsistent specifications; or further along the project's life-path the effects of poor planning and/or estimating may manifest themselves. In terms of project management, there may be no clear assignment of authority and responsibility, or on the other hand lack of adequate tools and techniques and not enough, or even wrong, user involvement may be the cause of what has gone wrong. The causes are many, complex, and they interact with each other. In some cases a single factor may spell out the problem whereas in others it is the combination of many small and apparently insignificant factors that are to blame.

**Everything is not always what it seems to be**

While the reasons for these failures are many and various; they are not always what might have been expected. The Arianne V rocket failed because its navigation software was inappropriate for the dynamics of that rocket design and, on experiencing questionable data instructed the rocket to self-destruct [9]. This was not actually a software failure, but one where the overall rocket system had been incorrectly assembled. In another case in 1999, during a NASA Mars projects, it was discovered that Imperial Units had been entered into the landing data where an appropriate metrical figure was expected – with disastrous results. In other instances, external influences have played a significant part such as the Denver Airport baggage handling saga [4], where an already confused project became inextricably embroiled within the local Political situation in Colorado.

What this means 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.

**Expectancy**

There is a fairly novel situation where public expectancy of computer technology is high and, when all goes according to plan, its contribution to modern lifestyles is most beneficial. This high expectancy fuels the competitive nature of computer. Within the more developed countries of the world safety-critical issues are more often than not handled by computers and once again the expectancy is extremely high and projects have to take account of this in their preparation. The result of all this means that the disciplines associated with the design specification and development of computer systems are almost held to ransom by their own success!

**Fashion**

"Fashion" has much to answer for. For example, the popularity of data warehousing has on occasions obscured its basic objectives [10]. Many efforts have failed because those involved were attempting to grapple with a concept that was huge or wholly inappropriate to their own particular circumstances. The causes of the problems may be due to size, but they are also due to the complicated and complex nature of the data warehousing concept itself.

**Stakeholder Interest**

Fashion can give rise to problems but another source of trouble is the varying nature of the interests of the stakeholders involved in a project. This often gives rise to conflict, frequently without any intentional criticism being involved.
In one United Kingdom Government project a very positive picture of a particular system is provided on their official web-site, [11]. On the other hand Hansard (the official record of proceedings in the House of Lords and in the House of Commons) tells a different story [12], [13].

Finally, on top of all of this is the reaction of human beings to what is involved. It appears that to the human mind, even in the business context, software is fun; it is contrary and sometimes appears to be miraculous. When it works it is wonderful and everybody wants something wonderful to happen now! Meeting the kind of expectancy generated by such emotions will never be an easy task.

## *Building a formal approach to coping with systems failure problems.*

In order to be able to suggest possible ways of preventing software failures, it is necessary to investigate in detail, what is going on in the real software world. To do this it is first necessary to examine what has gone before with a view to determining what general characteristics tend to become apparent. This has to be taken back over a number of years in order to give a wider sample of data. Such an approach also means that a retrospective look will determine if changes in available software, hardware, or in the change of rate-of-advance of those technologies, implies that what has gone before may be applied with confidence to the present day.

**The symptoms of systems failure**

The symptoms of failure can be grouped into groups which occur fairly commonly. For example:

- Technology
- Coping with Technology
- Requirements
- Insufficient Resources
- Malicious (Technical)
- Malicious (Industrial)
- Internal politics / Inertia
- Government policy
- Grandiosity

These may be split into (e.g.) "large projects" and "small projects" and just as one might find problems with scaling up software from small to large, the problems experienced with small and large projects may differ considerably. What appears as "large" to one organisation may appear as "small" when compared with (e.g.) a Government Department system or project. It is as yet

unclear just what the implications of size are, and just how the inferences drawn from one type of project may be used to seek solutions for similar projects on a different scale. On the other hand many of the problems experienced such as "Requirements Elicitation" are common to most projects regardless of size. These differences however have to be examined to find key causes. Other variations on the same theme include the size of the organisation, the type of technology being used and the degree of autonomy allowed to CIOs in each project.

## *3. The effects of Systems Failures*

Systems failures affect the organisations involved in creating, maintaining and using them and they can have a profound effect on the people involved, directly or indirectly.

The organisations affected cover the entire spectrum of business and service communities and where appropriate, those involved in any way with systems. (this therefore brings in the worlds of arts and entertainment) and the very significant personal use of computers experienced in the world today through (e.g.) e-commerce.

The people affected include those involved in developing or maintaining systems such as project/software managers, systems analysts and programmers, graphics specialists, operators and so on. But they also include other people connected to the systems or projects concerned such as directors, line managers, service personnel and other employees; together with other external suppliers, consultants and customers.

Turning to the effects on the organisations and people involved in Systems Failures; these may be profound. A number of definitions of the state of failed or failing systems are working their way into the lexicons of the software engineering and business communities. These include descriptions such as "Death March" for a project that has past the point where it becomes doomed; and "Runaway" where the time spent developing the system or the money spent on it have gone well beyond what is commercially viable or within the bounds of managerial common sense. Some typical features or such situations result in:

- Financial loss
- Depletion of assets and closures
- Job losses
- Lowering of moral ⇒ performance down
- Loss of shareholder confidence
- Bad press/media publicity … … ...

## 4. *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 much has been done to improve systems practice and management, a relatively small amount of research has been specifically devoted to addressing the subject of software systems failures.

Traditional well founded engineering disciplines have developed successful management practice over a very long time. It is true that mechanical and electrical faults will give rise to problems (and always will) but Software Systems were only been developed during the last half of the twentieth century and their culture is still immature [14]. In addition to this, 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.

Another facet of the problem is seen when it is seen that software development techniques have not matched the rate of advance of hardware technology. This rapidly changing technological scene has been accompanied by equally dramatic falls in prices of memory and data-storage media. 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 [15]. However this is not the whole story: it is essential to be able to recognise failure signs as soon as they start to emerge and be able to know what to do in a given failing situation.

Much has been written to describe the failures themselves [4] but the unfortunate fact is that not much research is going on in the area. Some exceptions to this may be found at NASA Goddard (SEL), with input from the University of Maryland, and similar work is being conducted at three USA universities funded by SERC. In Europe, work has so far been confined to Kaiserslauten (de) and Linköping (se) with new project work taking place at Middlesex University (uk). Other closely related work on technology transfer is also taking place at Carnegie Mellon University and the SPC in Washington in the USA; and at the University of York (uk). Safety critical issues being addressed in Newcastle (uk) and City University, London (uk); and a different angle on systems failures is also being studied by a number of Business Schools in the USA and UK. In an area about which so much has been written, this does not represent a lot of "work-in-progress".

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

It is essential to identify the true causes of why a system failed, is failing or has the potential to fail. Current work at this point is directed at finding a means of taking symptoms and "distilling" them into the true characteristics of a system problem. This is not a question of treating the case with suspicion as often errors creep in to the very best of practices owing to very understandable coincidences of a number of events. At the more extreme end however, it is not uncommon to encounter situations where (e.g.) an apparent symptom has been given as "software package inadequacy" but was actually due to the private agenda of an individual within the project team. In any case, rigorous treatment as opposed to "opinion" is required.

What this has meant is gathering extensive cases from the literature and conducting field work to gather information from industry during on-going system problem and success scenarios. This process is currently taking place and an appropriate body of knowledge is under construction in the light of what is being uncovered.

**A formal representation**

Having assembled the data into a manageable format, there is a whole range of possible ways to assist with understanding these symptoms. A number of approaches have been considered: they vary according to the degree of tightness that the specification/modelling technique takes.

(a) At the most formal end of the scale, Petri Nets [16] were initially considered They use a symbiosis of state- and event- oriented graphs, with tokens bound to the states involved to cope with (e.g.) concurrency. The drawback of this approach is however that while he state transitions described provide a valuable way of explaining what is going on, the esoteric nature of the subject of systems failures means that the definition of the states involves becomes ery involved with huge numbers of variables (and constants) to be taken into account.

(b) At the softer end of the modelling spectrum are appraoches such as the OPEN Method "Object-oriented Process, Environment and Notation" [17], which has relevance in describing systems failure scenarios, by having a meta-model framework that may be tailored to individual projects and addressing personal skills, organisational culture and requirements.

Grounded Theory allows models to be created early in the life-cycle of a development project which can establish information systems requirements, support non-sequential process models (such as prototyping), and be used to assist with the derivation of preliminary architectural components [18].

The three basic elements of grounded theory are concepts, categories and propositions. Concepts are the basic units of analysis since it is from conceptualisation of data, not the actual data *per se*, that theory is developed. Corbin and Strauss [19] state that in their opinion, theories cannot be built with actual incidents or activities as observed or reported; (i.e. they cannot be derived from "raw data."). The incidents, events, happenings are taken as, or analysed as, potential indicators of phenomena, which are thereby given conceptual labels.

The Grounded Theory method is particularly adept at handling voluminous qualitative data, it has specific rigorous procedures for reaching theoretical formulations from such data, and is usually expressed in natural language. The very nature of systems failures and the broad-based nature of Grounded Theory and ethnographic approaches means that these concepts are playing a significant role in the search for solutions to the problems described.

(c) In the "centre" of these approaches to modelling techniques is the increasingly popular UML [20], which is rapidly becoming a standard modelling technique within the field of Software Engineering. In UML, activity diagrams are able to express tasks in conjunction with information flows, enriched by operators for branching, forking and joining an activity graph. Work in other establishments is proceeding with a view to extending UML For example an amended version of UML has been created [21] to describe "Task Models for Co-operative Work". Here a system has been modelled, which refers to state relationships among activity elements. Task dynamics may be modelled, including temporal ordering of activities.

The relatively simple yet comprehensive approach used in UML together with its almost universal acceptance and consequent familiarity, makes it a highly appealing approach for formally handling systems failures scenarios.

## 6. A hypothetical mechanism (SSD/CASE TOOL)

A schematic representation of the intended failure analysis process is given in figure 1. Basically it involves being able to take a given situation and examine it against other similar situations and logical sets of rules designed to assess what is actually taking place. The vehicle for this is an inference engine with the ability to on draw supplied in summarised-data and textual formats.

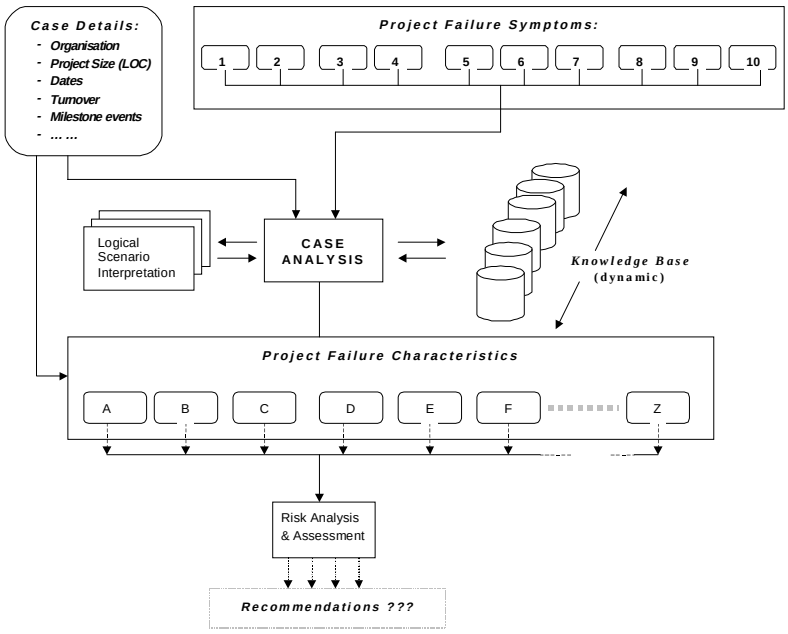

**Figure 1: A proposed approach to characterising systems failure symptoms**

## *7. Finding a means of preventing or solving software system failure a look at some existing methods.*

The foregoing discussion outlines measures currently being researched to accurately assess exactly what is going on in a particular situation, by looking at the raw symptoms and then logically distilling them into characteristics. A method for addressing how to rectify the implications of the results has to be sought.

The literature indicates many ways of handling software management that are both quantitative and qualitative Initially a study should be made of existing software quality maintenance and improvement techniques such as (e.g.) GQM [22] and CMM [23], which are methods designed to ensure success through good custom and practice within project management. These and other approaches will be considered, taking into account both qualitative and quantitative approaches. With systems failure in mind, the following points are noted:

**Goal-Question-Metrics (GQM)**

Many software metrics programmes have failed because they had poorly defined, or even non-existent objectives. The basic idea of GQM is simple and intuitive, with project management proceeding according to three stages:

1. Setting specific goals in terms of purpose, perspective and environment.
2. Refining these goals into quantifiable questions that are easy to understand.
3. Deriving the metrics and data to be collected, together with the means for collecting them, in order to answer the questions.

**The Capability Maturity Model (CMM)**

It will also be useful to consider the implications of CMM which describes the principles and practices underlying software process maturity. It is intended to help software organisations improve the maturity of their software processes in terms of an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes. It is organised into five maturity levels: Initial, Repeatable, Defined, Managed and Optimising.

Predictability, effectiveness, and control of an organisation's software processes are intended to improve as the organisation moves up these five levels. For Level 1, the software process is largely ad hoc. However, except for Level 1, each maturity level is decomposed into several key process areas that indicate the subjects that an organisation should focus on, to improve its software process.

**SPICE**

The SPICE project [24] was devoted to being able to provide an international standard that provides a framework for assessing the software process. Within a process improvement context, process assessment provides the means of characterizing the current practice within an organizational unit in terms of the capability of the selected processes. Analysis of the results in the light of the organization's business needs identifies strengths, weakness and risks inherent in the processes. This in turn leads to the ability to determine whether the processes are effective in achieving their goals, and to identify significant causes of poor quality, or over runs in time or cost. These provide the drivers for prioritizing improvements to processes.

Process capability determination is concerned with analysing the proposed capability of selected processes against a target process capability profile in order to identify the risks

involved in undertaking a project using the selected processes. The proposed capability may be based on the results of relevant previous process assessments, or may be based on an assessment carried out for the purpose of establishing the proposed capability.

## 7.2 Newer methods

A number of other approaches are being developed, which address how to assure system success. Three of these are briefly described below, as they are highly relevant to this project:

### The Experience Factory

The principle of the Experience Factory approach by Basili [25], is to learn 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. It consists of two organisations which separate project development concerns from organisational concerns of experience packaging and learning. The experience factory provides the processes and support for analysing, packaging and improving the organisation's fund of experience. The project organisation is structured to reuse this stored experience in its development efforts.

### Project De-escalation

Failing systems and runaways often attract additional effort and Project De-escalation [26] addresses this phenomenon. When de-escalation occurs, there is reduced commitment to a failing course of action, with the result that troubled projects may be successfully turned around or sensibly abandoned. It is essentially a business-oriented solution which appeals to the hard economics of the situation and the nature of the management involved. Some examples of its key actions are:

- Changes to top management
- Reducing managerial threats
- External shocks
- Going public on the problem
- New goals confirmed at high level
- Change of project champion
- Not punishing whistle blowing

#### Risk handling

In the rapidly changing field of software engineering, new ground is continually being covered or previously untried directions have to be contemplated. Software Risk Management, such as proposed by Boehm [27], takes a pragmatic approach to what is going on in an existing system or in a project. Its principles cover considerations such as: unrealistic schedules and budgets; shortfalls in performance; technical capability; and changes to requirements.

#### Focussing on the organisation

Another approach [28] involves strictly examining the implications of the managerial infrastructure in place and in examining the relationships between the various entities involved. The next task is to match up software engineering practices to that revised structure. Current practices in most organisations today is the opposite, where those involved seek to achieve best practices within an existing organisational set-up.

#### Business Process Re-engineering

## *Conclusion*

This project has taken the experiences from a previous Esprit software engineering projects (MutliSpace and Space-UFO) forward. These projects focused on software quality issues and then related them to the factors affecting software issues. Current work is examining real-life instances in industry and a methodology is being developed for coping with the results that are emerging. This multi-pronged attack on the subject means that the groundwork has been created for formally handling statistical data together with more esoteric data through the application of techniques such as Grounded Theory. Existing approaches are at the same time being examined to see how they may be harnessed in such a way that the most appropriate approach may be adopted as and when it is required.

## *References*


[1] Helga Drummond, "Escalation in Decision-Making : The Tragedy of Taurus", Oxford University Press, ISBN 0 198289 537, 1997

[2] NIC Series - NIRS2 & Pensions Professionals Update, http://www.inlandrevenue.gov.uk/nic/coeg-nirs2/coeg-nirs2.htm

[3] J. Carey, N Gross, O. Port and M.Stepanek, "Software Hell", Business Week, Dec. 6, 1999

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

[5] A.Doyle, "Moving Target: Software problems are destroying the worlds most advanced Air-Traffic Control Centre", Flight International, May 1997.

[6] Robert L. Glass, Software Runaways: Lessons learned from massive software project failures, Prentice Hall, ISBN 0-13-673443-X (1998)

[7] I. Stokes "The Soul of a Project", Project Manager Today, February 1995

[8] B. Boehm, "Anchoring the Software Process", IEEE Software, July 1996

[9] ESA/CNES "ARIANE 501 - Presentation of Inquiry Board report, ESA Press Release Nr 33-96 – Paris, 23 July 1996.

[10] R. Hackathorn, "A data warehouse does more than collect data. It reflects a valid and consistent image of the business", Byte Magazine; August 1997;

[11] NIC Series – NIRS2 & Pensions Professionals Update, Http://www.inlandrevenue.gov.uk/nic/coeg-nirs2/coeg-nirs2.htm

[12] Social Security Contributions Bill, Hansard (House of Lords), Debate 25 Jan 1999

[13] Social Security-Pensions (Computer Changeover**)** Hansard -Commons Written Answer 18 Jan 1999.

[14] J. Bach, "S.E. – Coming of age or reaching too far?", Proc 13th Conference on Software Engineering & Training, IEEE Computer Society Press, ISBN 0-7695-0423-X, 2000

[15] Geoffrey James, "IT Fiascoes and How To Avoid Them", Datamation Magazine, November 1997

[16] 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,

[17] I. Graham, B. Henderson-Sellers and Houman Younessi, "The OPEN Process Specification"1st Edition, ACM Press, ISBN 0-201-33133-0, 1997

[18] Galal, G. H., & McDonnell, J. T. (1997). Knowledge-Based Systems in Context: A Methodological Approach to the Qualitative Issues. AI and Society, 11(1-2), 104-121.

[19] Corbin, J., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative Sociology, 13, 3-21.

[20] Booch G., Rumbaugh J. and Jacobson I., "The Unified Modelling Language User Guide" Addison-Wesley, 1998

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

[22] R. van Solingen and E. Berghout, "The Goal/Question/Metric Method: a Practical Guide for Quality Improvement of Software Development", McGraw Hill, ISBN 007 709553 7, 1999

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

[24] [23] SPICE ISO/IEC Software Process assessment

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

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

[27] B.W. Boehm, "Software risk management: Principles and practices", IEEE Software, Vol 8(1) 1991.

[28] Doherty, N. F. & King, M., (1998) "The consideration of organizational issues during the systems development process: an empirical analysis", Behaviour & Information Technology, Vol. 17, No. 1, pp. 41-51.