Association for Information Systems
AIS Electronic Library (AISeL)

PACIS 2000 Proceedings

Pacific Asia Conference on Information Systems (PACIS)

December 2000

Shortcomings in Software Development Project Management: An Analysis of New Zealand Cases

John Paynter
University of Auckland

Daud Ahmed
University of Auckland

Shortcomings in Software Development Project Management: an analysis of New Zealand cases

John Paynter, M. Daud Ahmed
Department of Management Science and Information Systems
University of Auckland
j.paynter@ auckland.ac.nz

Abstract
Many software projects are not successfully completed or cannot meet the user requirements due to project management shortcomings. This paper considers well-publicised cases of IT failure in New Zealand (INCIS and Genesis). The reasons for their failure are analysed as pointers to lessons that can be learnt. Scope changes should be carefully analysed before acceptance and planning should be reviewed and updated regularly to avoid potential project problems. Risk management is to be done by earlier planning e.g. listing of at least top ten risks or threats to project. Project visibility needs to be increased for progress monitoring and reviewing.

Keywords: IT failure, project management, planning

1. Introduction

The failure of the Police Integrated National Crime Information System (INCIS) project is the most publicised one of many involving budget blowouts in the public sector (Jackson, 1999). Others include: the National Library's NDIS project and the Land Information New Zealand (LINZ) project. The private sector is not immune from similar failures, although they tend to be less well publicised. The INCIS project is reported to have cost between $100-200 million and Telecom's Customer Sales and Service project (CS&S or *Genesis*) is thought to be of a similar magnitude. The information in this paper is taken from internal and external project reports as well as interviews of some of those involved.

Software projects are inherently complex, and complex projects cannot succeed without careful planning. A well-planned project can be actively controlled, to do so you must ensure that progress is visible. Software projects are also inherently risky and they cannot succeed without active risk management (McConnel, 1998). Software Project management (SPM) is the first layer of the software engineering process (Pressman, 1994). As software is intangible, it is difficult to predict with certainty when a particular software process is likely to cause development problems. Software development projects are plagued by technical and managerial problems. More than 30% of all software projects in the United States are never completed, and of those completed, only 9-16% meet their time and budget projections. Over half of software projects exceed budget by 189% (anon, 1997).

Software development projects are different from other engineering projects. The success of a software project depends on how the project is managed. This paper focuses on the shortcomings in different project management activities and discusses possible corrective measures for these. Those covered are transfer of experience from previous projects, change of scope, planning (including estimation, scheduling, risk management, project control, progress monitoring and reviews) human resources management and teamwork, project managers and consultants, project visibility, work environment, user involvement and management tools.

2. Project Management

The project management body of knowledge defines project management as the application of knowledge, skills, tools and techniques to project activities in order to meet or exceed stakeholders' need and expectation from a project. Metcalf (1991) opined that project management embraces four basic disciplines: methodology, or procedures (15%), personnel management (50%), communications (25%), and planning techniques (10%). Some projects never seem to terminate . . . rather, they become like Moses, condemned to wander till the end of their days without seeing the promised land (Keider, 1974). While it is difficult to obtain statistics on the actual frequency of IT failures, various sources suggest that at least half of all IT projects are not as successful as we would like them to be (Gladden, 1982; Lyytinen and Hirschheim, 1987). While there are undoubtedly many different modes of IT failure, one pattern of failure that has been observed but seldom studied is the IT project that seems to take on a life of its own, "continuing to absorb valuable resources without ever reaching its objective" (Keider, 1974; Lyytinen and Hirschheim, 1987; Meredith, 1988). Eventually, these projects are abandoned (or significantly redirected), but the cost of having funded them can represent a tremendous waste of organisational resources (Keider, 1974).

Major modes of software project failure are late delivery or never completed, poor reliability, cost overrun and user dissatisfaction for exhibiting poor performance characteristics i.e. failing to meet the requirements (Brooks, 1975). The actual costs of software projects often greatly exceed the estimated cost. Other projects are completed within time and cost but do not provide full user satisfaction. The skillful integration of software technology, economics and human relations in the specific context of a software project is not easy. Poor strategic management and related human factors are the major causes of failure (Rodrigues and Williams, 1997). The main areas of software project management shortcoming that lead to software project failure are discussed below.

2.1 Transfer of Experience from Previous Projects

Effective project management depends on the use of experience from previous projects by both the project manager and the organisation. Efficient transfer of knowledge between projects is complex as each features unique characteristics. The management team may encounter problems in distinguishing the similarities and dissimilarities from among projects. Management techniques derived from small-scale projects may not scale up to large systems development (Sommerville, 1995). The software project manager needs to satisfy a variety of constituencies yet theory indicates good software project management should simultaneously be simple, specific, and general. A good theory should be able to explain why the project encountered problems and prescribe improved approaches that would have avoided the problem (Boehm and Ross, 1989). Failure to learn from mistakes has been a major obstacle to improving software project management (Abdel-Hamid and Madnick, 1990).

2.2 Change of Scope

A statement of software scope must be bounded (Pressman, 1994). Software scope describes function, performance, constraints, interfaces and reliability. Scope is related to business objectives and strategies of the organisation. Software project planning starts after bounding the scope. Due to the tendency for developers to add to the specifications, feature-creep, it is

equally important to specify what is outside the scope for a project (stage). The development time for a large software project may be many years; during that period the organisation may change the project objectives and requirements. Management may decide to stop the software development or to change the project to accommodate the changes to the organisation (Sommerville, 1995).

2.3 Planning

Planning is a management activity that deals with measurement techniques and estimation methods with risk analysis, scheduling and other decision-making activities. A Software Project Management Plan (IEEE 1058.1, 1987) describes objectives and sets constraints detailing project organisation, risk analysis, resource requirements, tasks, project schedule and monitoring and reporting mechanisms (Pressman, 1994; Sommerville, 1995). Software projects generally are staffed with technical people, who prefer technical work rather than planning. Many technical managers lack adequate training in technical management, so depend mostly on planning tools without the proper knowledge of planning. Failure to properly plan is one of the most critical mistakes a project manager can make. Thus the average project spends about 80 percent of its time on unplanned rework and fixing mistakes that were made earlier in the project, leading McConnel (1998) to argue that success in software development depends on making a carefully planned series of mistakes in order to avoid making unplanned large mistakes. Early planning is necessary to avoid doing extensive rework. While the macro-level aspects of project planning and control have been addressed extensively, research on the micro-empirical analysis of individual decision making behavior is lacking (Abdel-Hamid, et al., 1993). The initial assumptions, estimation and scheduling should be pessimistic rather than optimistic. There should be sufficient contingency built into the plan so that the project constraints and milestones need not be renegotiated every time in the planning cycle (Sommerville, 1995). Planning is an iterative process that is only complete when the project itself is complete.

2.3.1 Estimates

Project estimates provide a foundation for project plans. A careful estimate leads to scoping the project appropriately, which in turn leads to budgeting, staffing and scheduling it appropriately. A poor estimate can undercut the project in all these respects, making it difficult to complete the project successfully and impossible to complete it efficiently (McConnel, 1998). Poor estimation and poor visibility hamper software project planning and control (Abdel-Hamid, et al., 1993). Typically, a software project planner estimates the project as if nothing will go wrong and then increases that estimate to cover anticipated problems, with a further contingency factor to cover unanticipated problems (Sommerville, 1995). Project managers have to ask for funding for the entire project before they have had a chance to complete much exploratory work. Such requests inevitably miss the mark because too little is known about the requirements to support creation of meaningful cost and schedule estimates (McConnel, 1998). A software life cycle model is often used to divide a software project into phases that can be used as a basis for estimating the project cost in terms of time, money and people. The important factors in estimating software projects are: complexity-novelty, staff characteristics, clarity of project requirements, development environment, performance constraints, human resource availability, project size, support commitments, target audience, vacation, sickness, staff turnover and number of interruptions (Schroeder, 1991).

Estimates of the completed portion of development work increase as originally projected until a level of 80%-90% is reached. At that point, estimates increase slowly until the work actually is completed. Analysis of a NASA software project (Abdel-Hamid, 1988) indicated that the problem results from interaction are under-estimation and inaccurate measurement of progress due to poor visibility. To improve estimates, software producers need to develop and maintain databases (using tools such as COCOMO) of organisation-specific software project metrics such as cost, duration, error rate and staffing level. Software estimation can never be an exact science but a combination of good historical data and systematic techniques can improve estimation (Pressman, 1994).

2.3.2 Scheduling

A major problem in the classical life-cycle approach to software development is the completeness and clarity of the user requirements. Alternate paradigms, such as the use of prototype software models, may be timelier. Prototyping is shown to be an appropriate approach that can be used as a significant feature of the more formal life-cycle process, with little overall reduction in project control (Rowen, 1990). Limitations of scheduling include overlooking required tasks and the unavailability of key team members. A delay in one task causes cascading delays in dependent tasks. The schedule should provide more time than expected for unfamiliar aspects of the design and implementation (McConnel, 1998).

2.3.3 Risk Management

Risk management is crucial to good software management project, yet many projects are undertaken with no specific consideration of risk. Early identification of mistakes or external threats and prompt curative measures should minimise the risk. The project document should describe at least ten major risks of the project and the way the risks could be tracked and monitored. Gilb (1988) says, "if you don’t actively attack the risks on a software project, they will actively attack you". Failure to plan, failure to follow the plan that has been created and failure to revise the plan when project circumstances change are some of the most serious software project risks specifically related to planning (McConnel, 1998).

2.3.4 Project Control, Monitoring and Reviews

Software projects might be controlled to meet their schedule, budget and other targets. Management of changes to requirements is to be done carefully so that only necessary changes are accepted. A quality assurance plan that includes both technical reviews and testing assures that the project will not succumb to a costly, defect-ridden build-and-fix model. Progress monitoring and review is a control mechanism, identifying shortcomings and anticipated problems. Progress is compared to milestones and deliverables stipulated in the scheduling document. The PM-Net model (Lee, et al., 1994) may be used for representing and monitoring the software development process. This model provides information for progress management, as well as information of project status at different levels of detail, for the benefit of project managers. The model emphasises bottom-up data collection and top-down information inquiry functions. The data flow diagram (DFD) and work breakdown structure (WBS) techniques are used for construction of a hierarchical structure of the software development process. This process can be viewed as a set of activities, with each activity viewed as a set of sub-activities and each sub-activity as a set of tasks.

2.3.5 Project Visibility

The concept of visibility, referring to the ability to determine a project's true status, is closely related to project control. If you want good visibility, the project team has to plan it into the project from the start. Project reviews, unit development folders and computer-aided software engineering tools can be used to improve visibility (Abdel-Hamid, 1988). The Software Project Survival Guide (McConnel, 1998) recommends: regularly comparing actual performance against planned performance; using binary milestones to determine whether tasks are done; and revising estimates at the end of each phase.

2.4 Human Resources Management and Teamwork

Project managers are constrained in their choice of staff by factors of availability, budget, skill level (the best people may be otherwise allocated), difficulties and delays in recruiting and deployment of staff on projects to learn and gain experience. The industry's persistent managerial turnover and succession promote instability, leading to a discernible shift in cost/schedule trade-off choices affecting staff allocation and ultimately project performance in terms of both cost and duration (Abdel-Hamid, 1992).

Teamwork is an important feature of most software-development projects, but computerised project-management systems give little support for cooperation and communication among team members. Many programmers do not feel the need to work closely with other people. The majority of project management systems focus on planning and control at the top and intermediate levels, assisting project managers as far as coarse sub-tasks and activities are concerned, but not transforming these into schedules for individual team members (Kurbel, 1994). Project staff require extensive training in project teamwork. The key steps for the introduction of a comprehensive project management system are to create the environment, involve the team and achieve control (Metcalf, 1991).

2.5 Project Manager and Consultants

In today's software development environment, researchers, academics, engineers and managers are focusing more energy toward developing software engineering as a science in order to deal with the growing complexity of software applications. In so doing, there is a trend toward losing sight of the product, process and technology for which software is being developed. Bridging the gap between software and product development requires effective leadership and project management within the software design teams (Pulk, 1990).

Many software development and project management experts work independently as consultants, providing services on a full- or part-time contract basis. Such experts proffer multi-dimensional real-life experience in various fields of project management. They can assist the company in creating an environment in which it can continue the work and carry out the management of projects on the organisation's behalf (Metcalf, 1991).

2.6 Work Environment

Most software engineering takes place in environments designed for other functions, principally business offices. McCue (1978) concluded that the open plan architecture favored by many organisations was neither popular nor productive, encroaching on the important environmental factors of privacy, outside awareness and personalisation. DeMarco and Lister

(1987) found that programmers with good working conditions were more than twice as productive as equally skilled programmers in poorer conditions. Work areas should be congestion and noise free, with adequate light, airflow, etc. Software development is a deeply intellectual activity, bearing few interruptions (McConnel, 1998).

2.7 User Involvement

User involvement is critical to a software project in several respects, especially for user interface design. Success in building software hinges on building a product that end-users will use and like. Without end-user involvement, software developers are notorious for crafting technically elegant solutions to problems that users don't care about (McConnel, 1998). Large software development organisations have consciously insulated developers from the user. In a case study (Steven and Jonathan, 1994) Poltrock et al. mention that interface developers do not know how to build a friendly user interface because they don't understand the user. They argue that the interface developers should at least spend a couple of hours a year with the user to observe their usage pattern, so that, they understand the user requirement. Authors are concerned that the developers may be excessively influenced by the request for features from customers who may not be the representative of the market place (Steven and Jonathan, 1994). Gould (1988) in 'How to Design Usable Systems' recognises that most designers give some consideration to users but few involve users directly in the design process or plan for design iterations.

2.8 Management Tools

About 400 project management software packages are on the market, about one-third for personal computers. Most are oriented to generic project or time management. However some view software project management as specialised and that the tools should be highly designed to support specific tasks (Wylie, 1986). The Capability Maturity Model for Software (CMM) developed by the software community with stewardship by the Software Engineering Institute, provides the management tools to define a software development project. The methodology requires the tracking of cost, standardising the software process, measuring the software process and continually improving the process with quantitative feedback (Mandell, 1997).

3. INCIS

There is no suggestion that the non-transfer of experience from previous projects caused problems in the INCIS project. However the Police representatives on the project may have lacked the experience of the (IBM) contractors. The most publicised cause of the cost and time overruns was the change in the project's scope. One major change was from IBM's OS2 operating system to that of Windows NT. This change was costed at $12 million. Additional requirements to include traffic, firearms and family violence databases have an attributed cost of $14 million (Jackson, 1999). If there was a fault in the planning of the project it probably lies in the area of too many reviews by a variety of boards and committees (e.g., Treasury, Police). In the human resources arena, the State Services Commission identified a lack of strength in project and contract management skills. Although as the main contractor IBM could be expected to bring strong project management skills to bear on INCIS, one problem was that much of this was based offshore (in California) and conflicts occurred. This could conceivably lead to problems in the teamwork across the various sites and also between the Police and their contractors. IBM would have ample recourse to consultants within their organisation but as has

been highlighted above, this is not the case in the public sector. There was also the suggestion that there was disagreement on the project within IBM (Hutchinson, 1999).

No evidence suggests problems with estimation or scheduling although this might be revealed later in the official enquiry. It would appear from a review of the project that there was no risk management – a factor that would appear to be largely responsible for the failure of the project and the failure to take remedial action earlier. This is allied with the problems of the disparate project control and tied up with the fact that although there were (too) many reviews (more than 40), there was little project control. The fact that the phases of the project were so large would suggest that there was low project visibility. Nothing was reported about the work environment or management tools but it is unlikely that either contributed to the project failure. The Police Association did not receive the project favorably (not surprisingly when it was to be paid for by a lowered head count) and there was little user involvement by the rank-and-file. The Association called for the restructuring to be halted while INCIS was sorted out.

4. CS&S

There was low transfer of experience from previous projects as most people were brought in and largely isolated from existing Telecom projects. Unlike INCIS there were no large-scale changes in the project's scope but many small changes in the delivered project, e.g. Stage 14. These were not so much new features but annoying and counter-productive alterations to existing ones. However planning: suffered from indecision on the implementation platform (Plan A or Plan B – in any event the decision was deferred).

The mixture of human resources utilised caused problems. There was a hodge-podge of users, permanent employees, individual contractors and contracted organisations. Allied with this, at times it was difficult to get access to key users, or key development staff. The project suffered from poor teamwork – there was a low level of cooperation and distrust between teams (but not so much within teams) and projects. Although consultants were employed they were from many organisations and countries, bringing in different methodologies, techniques and work cultures. There was conflict between support staff trying to enforce (changing) methodologies and documentation, and the development teams. Conflict also arose with people reviewing other team's work and trying to integrate the various models against the corporate model.

Within stages the estimation and scheduling were unrealistic. The early setting of deadlines often resulted in throwing more people at a problem. Risk management was ostensibly ignored, although there was provision for this in some of the project documents, the sections went unfilled. Project facilitators (support staff) and managers were used for project control and utilised management tools such as Microsoft Project. There were comments that there were too many reviews and too much time spent in meetings, yet communication was lacking between projects and stages. Despite the project being divided into many deliverables (stages) project visibility was low on account of the (too) complex business model. The work environment did not seem to detract from the project although there was overcrowding. The lack of meeting rooms and personal space in which to try to think 'outside the square' from time-to-time did cause some problems as did frequent changes to the seating and reporting arrangements. User involvement was high during implementation but low prior to that. However it was commented that the project was signed off at too high a level – the senior managers lacked knowledge of operational requirements (Anon, 1999).

5. Conclusion

The backlog of software development projects is steadily increasing and cost overruns and schedule slippage are expensive, but methods used for cost estimation and project control are not standardised or reliable. Therefore, management requires a better understanding of the process, and a set of simple methods based on a sound, scientific foundation. Software project failure can be reduced to a greater extent by efficiently applying the project management techniques. A bounded software scope with subsequent changes analysed before incorporation into the revised scope. Planning is an iterative process and it should be reviewed and updated regularly. The initial planning assumptions should be pessimistic rather than optimistic so that any constraints can be easily overcome.

Projects need to be staffed with people having sufficient competence levels, as planned in the project document. Staff should be deployed timely and a project should not be considered as a training venue for the unskilled staff due to their toll on others' time. The knowledge, expertise and leadership style of the project manager have a big impact on the project. The project manager should be well trained in software development projects. In addition, the project can take assistance either from the individual consultants or management-consulting firms to strengthen weak areas in the development team.

Project managers must estimate costs, time and effort at an early, even exploratory stage, with sufficient contingency. Estimation tools are imperfect for specific projects. They should be used carefully and multiple methods should be used. Estimation may be made in two phases (preliminary and detailed) so that the detailed phase could consider most of the activities correctly. The project schedule shall clearly identify all the project activities and must allocate more time for unfamiliar activities.

Risk management is a critical activity of project management. Risk management is added by earlier planning including approaches such as listing ten risks or threats to the project. Project visibility is needed for controlling, monitoring and review of activities. All the milestones and deliverables must be clearly defined to ensure good project visibility. The work environment impacts the performance of engineers, so should nurture intellectual activities by being distraction-free and allowing sufficient privacy.
A user-friendly interface can enhance the user satisfaction. It is difficult to involve the user directly to the development process but interface developers can design a good interface by knowing user's usage pattern in the field.

At the conclusion of each project a debriefing should be held so that the lessons learnt from the success or failure of the project can be applied to future development.

References

Abdel-Hamid, T. K. "Understanding the "90% Syndrome," in Software Project Management: A Simulation-Based Case Study," *Journal of Systems & Software* (8:4), 1988, pp. 319-330.

Abdel-Hamid, T. K. "Investigating the impacts of managerial turnover/ succession on software project performance," *Journal of Management Information Systems* (9:2), 1992, pp. 127-144.

Abdel-Hamid, T. K. "A multiproject perspective of single-project dynamics," *Journal of Systems & Sofware* (22:3), 1993, pp. 151-165.

Abdel-Hamid, T. K., and Madnick, S. E. "The Elusive Silver Lining: How We Fail to Learn from Software Development Failures," *Sloan Management Review* (32:1), 1990, pp. 39-48.

Abdel-Hamid, T. K., Sengupta, K. and Ronan, D. "Software project control: An experimental investigation of judgment with fallible information," *IEEE Transactions on Software Engineering* (19:6), 1993, pp. 603-612.

anon *NZ Infotech Weekly*, 30 June 1997,

Boehm, B. W., and Ross, R. "Theory-W Software Project Management: Principles and Examples," *IEEE Transactions on Software Engineering* (15:7), 1989, pp. 902-916.

Brooks, F. *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading, MA, 1975.

COCOMO "COCOMO Model," http://www.softstarsystems.com/overview.htm#The Intermediate Model

DeMarco, T., and Lister, T. *Peopleware: Productive Projects and Teams*, Dorset House, New York, 1987.

Gilb, T. *Principles of Software Engineering Management*, Addison-Wesley, 1988.

Gladden, G.R. "Stop the Life-Cycle, I Want to Get Off," *ACM SIGSOFT Software Engineering Notes* (7:2), 1982, pp. 35-39.

Gould, J.D. "How to Design Usable Systems," In *Hand Book of Human-Computer Interaction*, 1988, pp. 757-789.

Hutchinson, M. "INCIS," *Managing Information Strategies, New Zealand*, (2:7), June 1999, pp. 25-28.

IEEE 1058.1 "Standard for Software Project Management Plans," *IEEE 1058.1*, Institute of Electrical and Electronic Engineers, Inc., 1987, Chapter 8.

Jackson, R. "Why IT projects go so horribly wrong," *Unlimited - business with imagination*, August 1999, pp. 54-60.

Keider, S.P. "Why Projects Fail," *Datamation* (20:12), 1974, pp. 53-55.

Kurbel, K. "Groupware extension for a software-project management system," *International Journal of Project Management* (12:4), 1994, pp. 222-229.

Lee, K.C., Lu, I.Y. and Lin, H. "PM-Net: A software project management representation model," *Information & Software Technology* (36:5), 1994, pp. 295-308.

Lyytinen, K. and Hirschheim, R. *Information Systems Failures-A Survey and Classification of the Empirical Literature*, Oxford University Press, Oxford, 1987.

Mandell, J. "Taming Chaos," *Software Magazine*, (17:8), July Supplement 1997, pp. 92.

McConnel, S.C. " *Software Project Survival Guide (SP*SG)," 1998.

[McCue, G.M., "IBM's Santa Teresa laboratory: architectural design for program development," *IBM Systems Journal* (17:1), 1978 pp. 4-25.

Meredith, J. "Project Monitoring For Early Termination," *Project Management Journal* (19:5), 1988, pp. 31-38.

Metcalf, B. "Software Project Management: The Role of the Consultant," *Industrial Management & Data Systems* (91:3), 1991, pp. 3-5.

Pressman, R.S. *Software Engineering - A Practitioner's Approach*, McGraw-Hill Book Company, 1994.

Pulk, B.E. "Improving Software Project Management," *Journal of Systems & Software* (13:3), 1990, pp. 231-235.

Rodrigues, A.G. and Williams, T.M. "System dynamics in software project management: Towards the development of a formal integrated framework," *European Journal of Information Systems* (6:1), 1997, pp. 55-66.

Rowen, R.B. "Software Project Management Under Incomplete and Ambiguous Specifications," *IEEE Transactions on Engineering Management* (37:1), 1990, pp. 10-21.

Schroeder, B.G. "Estimation Issues in Software Project Management," *Project Management Journal* (22:1), 1991, pp. 5-10.

Sommerville, I. *Software Engineering*, Addison-Wesley, 1995.

Steven, E.P. and Jonathan, G. "Interface Development in a Large Organisation: An Observational Study," *ACM Transactions on Computer-Human Interaction* (1:1), 1994, pp. 52-80.

Wylie, C. "Project Management: Beware the Thundering Herd -- More than 400 Software Products Are on the Market," *Computing Canada* (12:2), 1986,