RATIONAL *VS* SOFT MANAGEMENT IN COMPLEX SOFTWARE: LESSONS FROM FLIGHT SIMULATION

Mike Hobday, SPRU*
Tim Brady, CENTRIM**

May 1998

Paper prepared for submission to
International Journal of Innovation Management

Third Draft

***Science Policy Research Unit of University (SPRU) of Sussex**
****Centre for Research in Innovation Management (CENTRIM)**
of University of Brighton

Paper prepared for CENTRIM/SPRU/OU Project on Complex Product Systems, EPSRC UK Technology Management Initiative, GR/K/31756. The authors would like to thank Keith Pavitt, Jeff Rodrigues, Gillian Hardstone, Richard Nelson, Andrew Davies and staff at Dynamics for invaluable advice and assistance. The normal disclaimers apply.

**Science Policy Research Unit*
University of Sussex
Mantell Building, Falmer
Brighton, East Sussex BN1 9RF, England

Tel: 01273 686758/772726
Fax: 01273 685865
Email: P.A.Beeston@sussex.ac.uk

Abstract

This paper compares actual (A-type) software processes at work in flight simulation, one example of a software-intensive complex product system, with ideal, rational (B-type) processes as contained in company manuals, tools and procedures. The aim is to identify the causes and consequences of divergencies between A- and B-type processes in a complex product and draw implications for theory and practice. The paper also develops a simple partial model to show what A-type processes actually 'looks like' in practice. The evidence indicates that in response to industrial turbulence, uncertainty, technical complexity and difficulties in capturing user requirements, software engineers and project managers fall back heavily on A-type informal systems and 'soft' management become essential to project progress. Under such circumstances, B-type rational systems are inadequate to the task at hand and soft factors such as good will, negotiation skills, rule breaking and informal communications become essential to project success. These findings contrast with most approaches to software engineering which try to impose highly rational processes and tend to ignore soft issues. The paper also indicates potential benefits of the rational approach, sometimes overlooked by contemporary organisational scholars, arguing that it is the manner in which B-type processes are developed and implemented which leads to divergencies and difficulties, rather than the processes themselves which are an essential part of orderly progress. While the findings may not be relevant to simpler products and simpler tasks, other complex product systems may well face similar divergencies and project management challenges.

INTRODUCTION

This aim of this paper is to analyse the software processes involved in designing and producing a flight simulator, an example of a software-intensive 'complex product system' (CoPS).[1] The focus is on the difference between how software development is *actually* undertaken, compared with views, models and recommendations on how the process *should* proceed. The paper introduces a simple method of identifying actual (A-type) *vs* should be, rational (B-type) practices in order to identify the nature of the differences between A and B, examine their causes and consequences and draw implications for theory and management practice.[2]

The research motivation is both to improve understanding and contribute to the practice of software development in CoPS. As Section 1.1 shows, the software challenge in CoPS is particularly daunting because of design and production complexity, the difficulties in integrating large numbers of sub-systems, and the wide varieties of knowledge and skills involved in design and production. Because of the lack of certainty on how to proceed from stage to stage and the frequency of unexpected events, the gap between A and B is often high, leading to major problems or 'hot spots' in design, manufacture, delivery and cost control in CoPS.

The literature review (Sections 1.1 and 1.2) shows that, despite the recognition of divergence between A- and B-type practices in the software and organisational behaviour research fields, very little attention is given to the causes and consequences of such divergencies. The overwhelmingly dominant approach in software is to try and impose rational B-type practices, regardless of the informal realities which lie behind the formal procedures and systems. Indeed, the denial of A-type practices is sometimes seen as a virtue in software research and engineering practice (Parnas and Clements, 1986).

The research approach (Section 1.4) is to examine in detail one particular project ('Triumph') in one major supplier ('Dynamics'), as an illustrative CoPS case example.[3] The research method, developed with Dynamics as a standard tool for process analysis/process improvement, involves: (a) collection of data on rational, B-type practices as enshrined in toolkits, manuals, formal procedures and the views of senior managers; (b) identification of real, A-type practices from in-depth interviews with software practitioners and project managers; (c) comparing A- and B-type practices for divergencies first in interviews and then in practitioner workshops; and (d) workshops to verify the causes and consequences of problems and to explore solutions to them. From a methodological perspective, the paper

---

[1] Defined below (Section 1.1). The study follows on from research on innovation in the flight simulation industry over several years. See Miller et al. (1995) for an analysis of flight simulation industrial structure and the role of regulators, users and specialist suppliers in the innovation process.

[2] In the organisation, management and software fields various terms are associated with B-type processes including rational, formal, should be, normative, scientific, canonical, espoused, official and mandated. Conversely, A-type practices are associated with terms such as soft, informal, real, actual and non-canonical. Although these terms differ somewhat in origin and orientation, for purposes of simplification the terms A- and B-type are used throughout the paper (Sections 1.2 and 1.3).

[3] The name of the company and the project have been changed for confidentiality reasons. Section 1.4 explains the choice of this particular project.

attempts to exploit the existence of the gap between A-and B-type practices, first as a means of intra-project benchmarking and second as a mechanism of feeding back data in real-time for the purposes of improvement.

Analytically the paper hopes to contribute to the software and organisational development fields by asking: (1) what, if any, are the divergencies between A- and B-type practices in a CoPS?; (2) do the gaps matter and, if so, why?; (3) what are the causes of the differences?; (4) how can problems caused by gaps be overcome (i.e. how can the gap be narrowed in practice)?; and (5) what do real, A-type practices actually 'look like' in a CoPS? The paper also hopes to bring together insights from the, normally separate, software and organisational development fields in order to assess the disadvantages and merits of the dominant, rational approach to software development.

As in any study of this kind it is important to draw lessons and generalisations very carefully as all projects differ to some extent, as do the environments in which they are undertaken (Section 5.2). Another limitation is the nature of the data. Although verified by practitioners in structured workshops, much of the data is attitudinal and qualitative. However, the purpose is not to provide an entirely rich case narrative (Orr, 1987; 1990) but to develop a broad understanding of major problems and possible solutions to them, for the purposes of improvement.

PART 1: ANALYTICAL PERSPECTIVES AND METHOD

This section explains the purpose of studying CoPS and points to the increasing importance of software in CoPS projects (Section 1.1). It then presents relevant organisational behaviour and management perspectives, focusing on the distinction between A- and B-type practices (1.2). Analysis of the software literature (1.3) shows the dominance of the rational B-type approach, while the method section (1.4) shows how the A *vs* B gap can be exploited for analysis and improvement purposes.

1.1 CoPS and software

CoPS can be defined as high cost, products, systems, networks and constructs which embody large numbers of tailored components and various knowledge and skill inputs. Examples include flight simulators, telecommunications exchanges, business information networks, high speed train engines, air traffic control systems, avionics systems, intelligent buildings, baggage handling systems, oil drilling equipment, integrated mail processing systems, printing machinery, hospital information systems and many other high value business-to-business capital goods.

Although there are many different categories of CoPS, they have certain features in common when compared with simpler, lower cost, mass produced goods (Hobday, 1998). Because CoPS embody many tailored components and sub-systems, product architectures tend to be very elaborate and the mastery of complex sub-system interfaces is often crucial to design and development, as is the integration of a wide variety of knowledge and skills. As a result, unit production is long in duration (sometimes several years) and usually carried out in projects or small batches. CoPS are normally tailor-made for particular customers and tend to be design- rather than manufacturing-intensive.

As industrial artifacts, the composition of CoPS often confers extreme work task complexity, uncertainty and difficulty. Production stages are often lengthy, messy and ill-defined leading to 'trouble shooting' as a key engineering skill. As a result, CoPS lend themselves to project-based, or matrix organisational forms (Woodward, 1958) or organic rather than mechanistic organisation, to use the words of Burns and Stalker (1961). Furthermore, because CoPS are critical to the performance of businesses, users and buyers (e.g. major airlines) often involve themselves directly in innovation and production processes. Production tends to involve a heavy emphasis on systems integration and project management, rather than the divisible, repeatable and measurable tasks more common in mass produced, simpler goods.

Over the past two decades or so, the diffusion of embedded software has improved the control, flexibility and performance of many CoPS, while mastery of software development, systems engineering and integration has become central to the ways in which CoPS are produced. For example, in flight simulation the concurrent design and production of major system components is usually carried out in software using predicted data and complex models.

The tasks of understanding, managing and improving software processes pose particular difficulties in CoPS. There is abundant evidence that software is a major stumbling block to the execution of many CoPS projects, leading to delays, cancellations and cost overruns (see Section 1.3 below). Software has accentuated the human, craft-based elements of CoPS design and manufacture, making the control of costs and time ever more difficult.

As discussed below, efforts to impose strict, rational B-type processes in CoPS are common. As a result there are often major differences between B-type formal procedures and A-type actual practices, as designers and engineers respond to unexpected technical difficulties, unpredictable system properties and emerging new customer requirements. Reconciling the gap between A- and B-type practices is a major challenge for CoPS producers and for organisational theory.

1.2 Organisational and management perspectives

If we turn to the contemporary field of organisational behaviour (OB), as Seely Brown and Daguid (1991) point out, very little attention is given to the implications of the A *vs* B gap (which they term non-canonical and canonical, respectively) for work, learning and innovation. Building on the work of Orr (1987, 1990) they show that the A *vs* B gap challenges conventional OB thinking and deepens the practical problem of improving organisational effectiveness. Indeed, Orr's evidence shows that attempts to impose B-type practices forces practitioners to mask their real activities, driving real work practices (as well as learning and innovation) underground, leading to an ever widening gap between B-type formal company policies, values and procedures and real A-type behaviours. As a result, sometimes B-type job descriptions, manuals, organisational charts, training programmes, tools, methods and espoused values of management become ridiculed by intelligent shop floor practitioners as bureaucratic, unhelpful and detached from reality.

Although the above authors tend not to distinguish between different classes of product and task, their findings are based on problem-solving in photocopier maintenance, a moderately

complex task involving personal initiative, re-interpretation of problems, informal action and user-interaction of a kind rarely captured - and often outlawed - in B-type procedures, manuals and tools. In fact, as they show, in "breaking the rules to get the job done", important organisational learning and innovation occurs, centred on real A-type work practices. Given the intrinsic task complexity and uncertainty involved in CoPS production, these findings are especially relevant to this study.

From an historical perspective, many management scholars have long recognised the A *vs* B distinction, arguing that the B-type rational approach is only appropriate for stable environments and relatively simple tasks which lend themselves to centralised, hierarchical organisational forms (or 'machine like bureaucracies' as Morgan, 1986 p56 puts it). Burns and Stalker (1961) show how the most appropriate organisational form, ranging from organic to mechanistic, is contingent on the task at hand. Similarly, Woodward (1958) shows that the project or matrix form is best suited to complex, one off production tasks. Mintzberg (1979) argues that 'adhocracy' is best suited to perform complex and uncertain tasks in turbulent environments. As Morgan (1996, p57) notes, the adhocracy usually involves project teams which are constituted to carry out a particular task and then disband when the work is completed. In CoPS, the project team often involves members from several different firms, further complicating the organisational dynamics.

Management theorists have not only shown that organisational form is contingent on the task at hand, but also that in all organisations an informal A-type reality exists which is central to performance, efficiency and effectiveness. Indeed, as early as 1918, the originator of scientific management Frederick W. Taylor (1911) was taken to task by Mary Parker Follett (then a member of the Taylor society) who argued that firms were social as well as economic units and that human relations and motivations could not be separated from operating tasks.[4] In the classic 'Functions of the Executive', Chester Barnard (1938) criticised Taylor and other rationalists, showing the importance and functioning of informal organisations, and their significance within formal organisations (Chapter IX). As Barnard showed, although informal practices are often overlooked or neglected, they are central to organisational communication and cohesion, as well as the personal integrity of practitioners.

Many other scholars have challenged the rational approach. Herbert Simon criticised the rational decision making view of humans, pointing out that actual decisions are taken under conditions of bounded rationality, involving perception, judgement and intuition (Simon, 1955). At the organisational level, Lindblom (1959) argued that under conditions of uncertainty and incomplete information, 'muddling through' in a step-by-step manner is the only rational way to proceed.[5] Writing in the same era, Klein and Meckling (1958), Marshak (1962) and Klein (1962) made similar points in their research into US military systems, concluding that, under conditions of uncertainty the decision making process involves substantial learning and a progressive narrowing of options, very different from the 'normal' production task of scheduling and resource allocation. More recently, Hilmer and Donaldson

---

[4] The debate between Taylor and Parker Follett (1918) is discussed by Peter Drucker in Graham (1995, pp24-31).

[5] Lindblom's insights were later developed into 'Logical Incrementalism' by Quinn (1980).

(1996) show why management systems need a soft type-A accompaniment of practical guidance and informal human support.

Despite such findings, many management thinkers and practitioners continue to demonstrate a strong desire to treat organisations as if they were rational. A contemporary version of the rational view is embodied in business process re-engineering (Hammer and Champy, 1993; Davenport, 1993; 1996). Its recent popularity in the US and Europe shows that the scientific, rational approach continues to run deep in theory and practice.

In CoPS, B-type tools and systems can emerge which are diametrically opposed to real A-type practices. As Sapolsky (1972, pp94-130) showed in his study of the US Polaris missile system, 'PERT' (Programme Evaluation and Review Technique), which was developed within Polaris, was not actually used to build the system and, despite its subsequent proliferation, was a deeply flawed management tool. Instead, Polaris's success was the result of inspired leadership, good management and a shared spirit of commitment across the organisations involved. PERT, according to Sapolsky, was a complicated-to-use, complex computer system used primarily to impress visitors (and US Government funders) to justify the programme and build up a myth of managerial effectiveness. In this interpretation, PERT was a deliberate B-type smokescreen, masking A-type real practices.

Given the above findings, an intriguing question is why have such insights failed to percolate through to a wide audience of theorists and practitioners? One possible answer is that despite the recognition of A *vs* B in a few specialist quarters, not a great deal of direct attention is given to the issue, as argued by Seely Brown and Daguid (1991). In a review of the literatures, Huber (1991) argues that the failure of OB to reach practitioners is partly because the field lacks cohesion and partly because very little OB is presented in forms and formats which give it 'social or administrative value' (p125). Another possible reason is that the rational approach does indeed have positive value in organic CoPS settings, a possibility often ruled out by its critics (a theme returned to in the conclusion).

1.3 The software engineering field

Software engineering in CoPS brings the rational *vs* soft debate into sharp focus. The very term software engineering implies that software has progressed to a systematic and predictable discipline. Large programmes of research (e.g. on metrics, structured programming, formal methods, computer aided tools and integrated project support environments) are all attempts to devise ways of developing software programs systematically and scientifically from a precise *a priori* statement of requirements.

Despite huge efforts to rationalise software, there is abundant evidence that software remains a (perhaps the) major stumbling block to the successful execution of major CoPS projects Gibbs (1994), Peltu (1992), Littlewood and Strigini (1992), Flowers (1996), Paulk, (1995), Collins and Bicknell (1997). In software, the term complex usually indicates a large and ambitious project with a high risk of designing the user interface incorrectly (perhaps due to unclear or changing customer needs), stringent performance requirements (including a low presence of errors in the operational task of the software), a high risk of incorrect budgeting and scheduling, and major systems integration challenges (Boehm 1988, p69, 71).

The term rational, which has a specific meaning in software, can be defined as the systematic development of a programme from a precise initial statement of requirements in a reasoned, logical and documented manner (Parnas and Clements, 1986). In the company discussed below, as in many others, B-type procedures are captured in company manuals and various process flow charts used to guide engineers.

In software the rational approach dominates and it appears that few, if any, of the OB insights have informed software thinking. However, the importance of A-type issues such as organisational learning, informal processes, leadership, team building and conflict (Senge, 1990; Leonard-Barton, 1988; Garvin, 1993; Stata, 1989) presumably apply as much to software as they do to any other domain of production. Leonard-Barton (1992), for example, shows how major new projects can threaten existing interest group interests and core capabilities in firms and how, by the same token, tried and tested methods can become 'core rigidities' in a rapidly changing market and technology environments.

Software tends to be analysed within its own, engineering-centred domain and many attempts have been made to improve software performance by modelling and recommending rational processes and procedures, regardless of the real working environment. Among the most widely cited prescriptive models are those of Boehm who introduced both the waterfall and spiral models of software development (1988, 1989 and 1991). DeMarco (1979) and Yourdon (1978) provide early reports on how to control software through formalised design reviews (called code walkthroughs) within firms. Abdel-Hamid and Madnick (1991) analyse software productivity and reflect on common mistakes made in software project management. Smith (1994) analyses the causes of software failures from a quality and reliability perspective. Humphrey (1995) examines how individual software practitioners might improve their engineering skills, while Culver-Lozo (1995) assesses the difficulties in scaling up best practices from small to large software projects. Kellner (1996) provides a formal method of software process analysis, based on computer simulations, quantitative analysis and predicted estimates of impacts of changes. Similarly, Pulford et al (1996) provide a purely quantitative approach to managing software. Some authors tackle the issue of how to integrate software into complex products and systems, again from a B-type perspective (Humphrey, 1989; Paulk, 1993; Boehm, 1988; Raghavan and Chand, 1989; Buxton and Malcom, 1991).

Some authors express concerns over some aspects of the rational approach. For example, Emam and Madhavji (1995) question the reliability of numerical measures of organisational maturity, while Baker and Rouse (1995) point to the dangers of over formalistic adoption of ISO9001 for software, especially for smaller firms. However, most of the field tends to ignore soft A-type issues, focusing instead on the B-type should be processes. Indeed, the Software Engineering Institute's (SEI) capability maturity model (CMM, see Annex 1), now an internationally accepted approach to quality standards, assumes (a) that organisations mature through various stages, towards rational, controllable processes and (b) that this progress can be measured and assessed (Paulk, 1995; Rout, 1995). The fully controllable rational process (Level 5 CMM) is the holy grail of software quality.

Of course, many software analysts realise that theory deviates from practice. However, the response to problems is normally to strengthen and impose more rigid B-type controls, formal systems and procedures (Jones, 1996), rather than recognise and analyse the gap between A

and B and consider its causes and consequences. In fact, some leading software analysts accept that the design and development of software is not and cannot be rational, but go on to claim that great benefits can be gained if you proceed 'as if' the process was rational (Parnas and Clements, 1986; Paulk, 1995a).

Over the years, problems of software management has led to unease among many leading software writers. In a series of review articles on large scale, complex software projects published in IEEE Software, two highly respected writers on software conclude: 'If past is prologue...we might expect the following: ever more concern with building healthy corporate culture; increasing awareness of teams and workgroups as sociological entities; continuing emphasis on good methods, controls and communications; and finally perhaps even a bit of Zen.' (DeMarco and Miller 1996 p27).

It is fair to conclude not only that OB and management studies have failed to inform the software field but also that, despite concerns over many project failures, the software profession retains a highly rational approach to work processes. The dominant software approach is consistent with the Taylorist scientific approach, which is applied even (and perhaps especially) in the uncertain, organic settings of CoPS. Curiously, while B-type rational processes underpin the dominant CMM approach to software quality, there is an underlying current of concern for human beings and real group dynamics.

1.4 Method - applying organisation development techniques to software processes

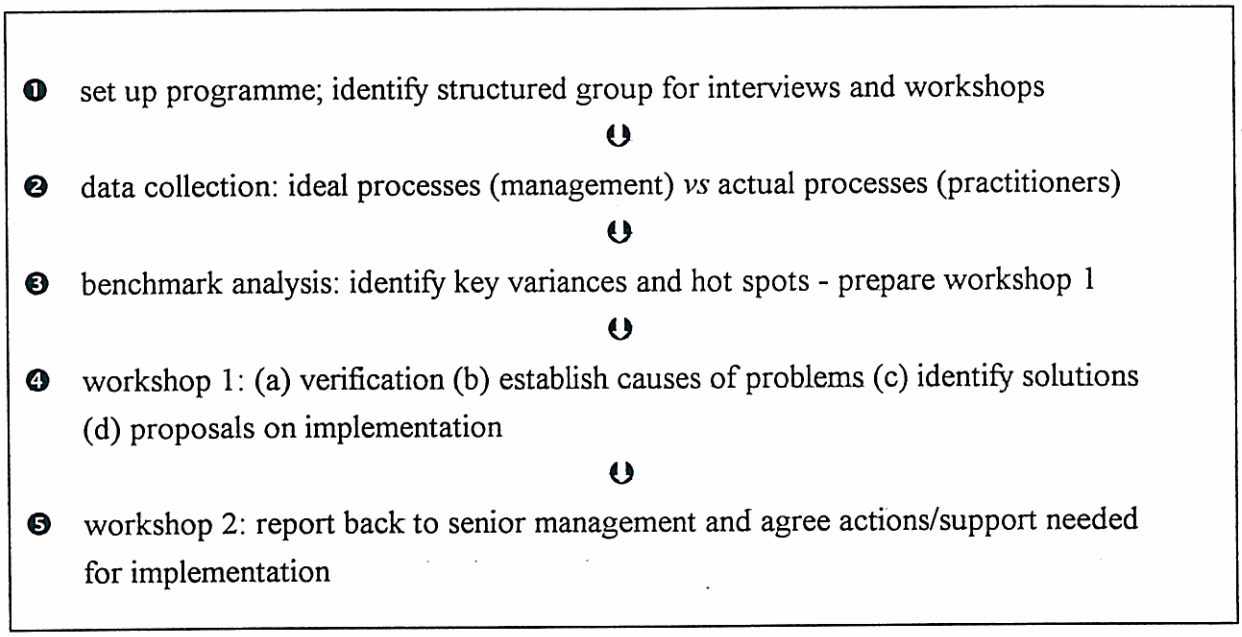
The study involved developing a real time method for analysing A and B-type practices in order to feed back information to Dynamics for verification, analysis and improvement purposes. The approach was to analyse one project in depth to gain detailed empirical insights and then to verify the accuracy and generalisability of findings in structured workshops with a sample of software practitioners (Hobday and Brady, 1997).

The approach is based on intervention techniques from the field of organisation development, a sub-field of OB. The tradition of organisation development, which spans both management strategy (Mintzberg, 1989) and implementation (Mullins, 1994; Handy, 1993; French and Bell, 1973; Tyson and Jackson, 1992) tends to treat strategy, management and work as iterative processes which are 'crafted', informal and sensitive to organisational style and human motivation, rather than top-down, scientific or rational. These authors show how selective outside interventions can sometimes be helpful in surfacing issues, identifying problems and stimulating new working practices in firms (French and Bell, 1973). Underpinning this approach is the belief that dealing with A-type informal, human processes is essential to organisational improvement. The method involved applying some of the lessons learned from organisation development to the issue of complex software development.

The method, described in detail in Hobday and Brady (1997), involved five basic steps, each with more detailed sub-processes and outputs. Step 1 involved agreeing with management and practitioners the scope, aims, outputs and timing of the exercise, and identifying a structured group of interviewees. Step 2 involved collection of data on rational, B-type practices as contained in toolkits, manuals, flow charts, formal procedures and interviews with senior managers on how the process should proceed. Process analysts at Dynamics contributed to the questionnaire design and to capturing, diagramatically, the formal company

processes. Data was then gathered from practitioners on how the process actually proceeded in project Triumph, using a structured questionnaire. Step 3 involved comparing A- and B-type practices for divergencies using the data, and generating a list of major 'hot spots' (problem areas) and 'beauty spots' (solutions and best practices). In Step 4 (workshop 1) the findings were presented back for verification, analysis of causes of hot spots, discussion of possible solutions to problems and proposals for implementation. Step 5 involved reporting back to senior management to feed back findings and agree an implementation plan.

Figure 1: Software Analysis and Improvement Method: Five Basic Steps



Interviews occurred in three phases during the project: shortly after the start in 1995; during the main period of design in 1996; and just after the project finished in 1997. A total of 40 interviews were carried out, involving 28 individuals, at most levels of seniority across all relevant departments.

The rationale for chosing flight simulation and project Triumph was as follows. First, flight simulation provides an example of a CoPS, involving highly complex software engineering tasks, several supply organisations and the user in design and development. Second, as a safety critical area, the software is subject to a great deal of formalised B-type, systematic controls, and specific management tools are utilised to control processes in order to ensure a high quality product. Third, the sample project Triumph was chosen because: (a) it was a significant project, just at the start of its development and could be tracked in real time; (b) as a military project, B-type tools and procedures were mandated and could be relatively easily identified; and (c) the project, although challenging, was fairly typical in that it contained a moderate degree of technical complexity and novelty, and an average involvement of outside suppliers and users. In the event, during the project major efforts were made to deliver a

rational software process to conform to UK Ministry of Defence (the user) procedures and documentation requirement.[6]

PART 2: HOW FLIGHT SIMULATION DEVELOPMENT SHOULD PROCEED

2.1 A complex product system

Figure 2 provides a description of a typical civil flight simulator. The main components and sub-systems include the image generator, host computer, cockpit, visual display, platform, motion system, cabin, instructor station, projectors, back projection screen, off simulator electronics, training facilities and other components. The trainee pilot sits in the flight simulator cockpit and 'flies' the aircraft to a destination entered into the computer, rather like a real aircraft. Unlike a real aircraft, an instructor is able to monitor and test the pilot as he or she simulates dangerous flight events such as engine failures. Today, most pilots receive most of their training on simulators for reasons of cost and efficiency. Some civilian pilots obtain a licence without ever having flown a real aircraft (so-called zero flight time training).

Most of the flight simulator functioning depends on software, involving complex models and existing or predicted data on flight behaviour and performance. The latter (so-called data packages), generally supplied by the aircraft makers, are central to the building of the flight simulator. Software development occurs in the normal way with the capturing of general customer requirements, followed by a series of more detailed design stages, coding, testing, hardware-software integration and maintenance (see Parts 3 and 4 below).

2.2 Stages of development in a 'rational' world

During our research we identified B-type formal stages of the building of a typical simulator in Dynamics ('what should occur') and attributed average times to each phase (see Table 1). A flight simulator project takes on average two years to complete. Most of the stages involve software activities of one kind or another, ranging from systems analysis, engineering, code writing, testing and systems integration. Special software skills are needed for avionics, computer generated imagery, flight performance (e.g. aerodynamic modelling) and systems interfacing. In large firms like Dynamics, engineering is supported by R&D into human factors (e.g. ergonomics and psychology), new hardware systems (e.g. special printed circuit boards, microprocessors and multimedia interfaces), flight environments and many other fields.

2.3 Skills and knowledge required

In flight simulators, as in many other embedded real time software systems, the required knowledge bases are varied and complex. Flight simulator producers are required to master at least four difficult knowledge and skill sets:

---

[6] Note that a second civil project, not reported on here for space reasons, was also analysed for purposes of comparison.

Figure 2: Description of civil full-flight simulator

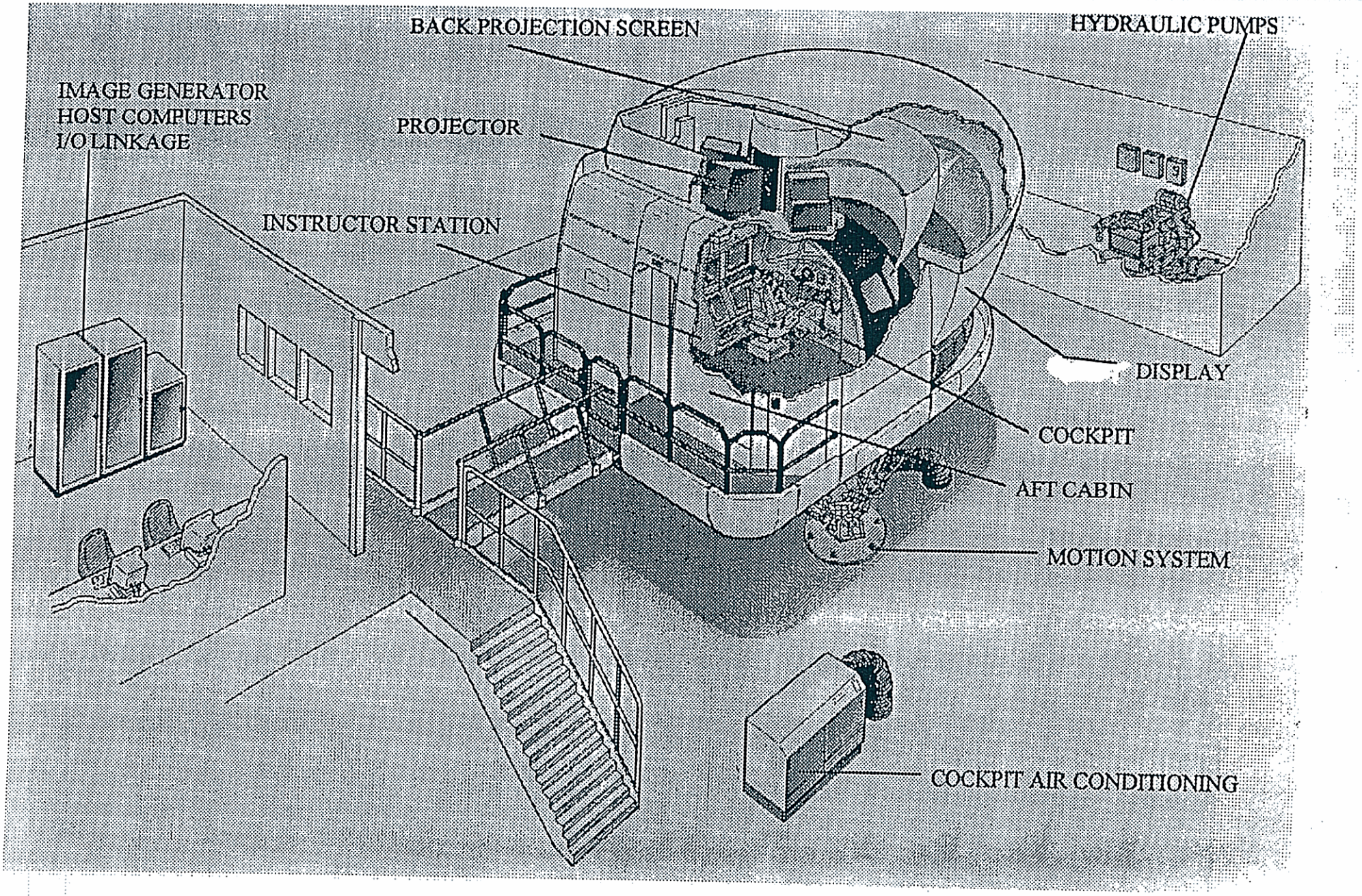


Table 1: Typical Life Cycle Stages of a Flight Simulator Project (Civil and Military)

Key stages	*Duration**
BID STAGES	
1. price bid in response to RFP**	4 weeks civil (16 weeks military)
2. negotiations with the buyer	1 week civil to 52 weeks (military)
DEVELOPMENT STAGES	
1. programme launch and communications	up to 8 weeks
2. systems requirements analysis	
3. provisional design work	
4. detailed design specification	
5. software structuring and coding stages 2 to 5	26 weeks civil; up to 52 weeks military
6. power-on phase	6 weeks
7. hardware commissioning	6 weeks from power phase
8. hardware/software integration	8 to 12 weeks civil (24 weeks military)
POST INTEGRATION STAGES:	
1. quality Testing (in factory)	6 to 12 weeks (military only)
2. customer testing (in factory) (factory acceptance certification)	3 to 6 weeks
3. break down and shipping	3 to 6 weeks
4. re-installation and re-commissioning on customer premises	6 weeks
5. customer acceptance on site (regulator certification)	2 to 4 weeks
6. the full-flight simulator is ready for training (RFT)	
POST-DELIVERY ACTIVITIES	
one to five year's warranty and service	
post-delivery support and services (up to 20 years life)	
spare parts delivery	

* several stages of military systems take longer due to customisation, performance and testing requirements

** request for proposal

Source: compiled from interviews at Dynamics in 1995 and 1996.

(1) the skills to integrate hardware and software components (motion, visual, computer and cockpit) into a coherent whole (the simulator);
(2) the know-how to use and develop the mathematical simulations which replicate the behaviour of the aircraft;
(3) the detailed knowledge of client requirements for training, checking and pilot quality programmes;
(4) precise knowledge of the rules and regulations (especially the acceptance test guides) which specify the requirements for simulator approval.

2.4 Core technical competence of the flight simulator producer

In recent years the core technical competence[7] of the modern flight simulator producer has become one of systems integration. With increasing complexity and technological advance, many of the sub-systems previously built in-house by flight simulator producers are today contracted out to specialist suppliers. These include visual systems from Evans and Sutherland, the computers which drive the flight simulator (e.g. from CCC and Silicon Graphics) and several other key components. While other major suppliers (and aircraft makers) can produce key parts of the simulator, the one major activity the flight simulator producer can perform (which others would find extremely difficult to imitate) is the integration of the various parts of the system.

Systems integration operates at two main levels: knowledge and physical components. At the component level, companies such as Dynamics take the aircraft maker's data package, as well as parts and sub-systems produced in-house and by the specialist suppliers, and combine them together using engineering and integration skills, involving each of the main hardware and software domains described in 2.1 above.

At the level of knowledge and experience, the capability to integrate the various sets of engineering, regulatory and training knowledge is needed to produce a flight simulator which delivers the 'feel' of the real aircraft. Systems knowhow enables the engineers to convincingly mimic the behaviour that human pilots expect from an aircraft and also to meet the regulators' safety requirements. Many of these skills are tacit, embodied in specialists who use their judgement and experience to approximate a convincing replica of the real aircraft. Human pilots play an important part in advising the software and systems engineers, not only at the early stage of requirements capture but also during the latter stages of final testing, revisions and trial. Despite the apparently technical and codified environment, producing a faithful replica of a real aircraft involves a substantial craft input and relies on highly skilled individuals in subject domains such as fuel systems, avionics operating systems, performance modelling and visual technology.

PART 3: THE TRIUMPH PROJECT

3.1 Project characteristics

Although Triumph was a fairly typical project it had some unusual features and posed some technical difficulties, as do most flight simulators. The customer, the Ministry of Defence (MoD), took a direct role in Triumph insisting on formal systems engineering tools and procedures.[8] There were several technical challenges (see below), notably the use of ADA, a

---

[7] For our purposes we confine the discussion to technical competence, rather than the broader business definition of core competence put forward by Hamel and Prahalad (1994, pp199-209).

[8] Systems engineering, like software engineering, can be viewed as an effort to achieve a rational production process. The systems engineering discipline originated in the 1950s and 1960s within the US Department of Defense (DoD) major military and space programmes (Sage, 1981; Shenhar, 1994; Boardman, 1990). It provides a highly technical, formal, engineering-based approach for the management and integration of large

highly structured programming language developed for real-time, embedded systems (which contrasts with the more widely used FORTRAN language).[9]

The Triumph flight simulator, developed for military pilot training purposes, was to mirror the operation of a modified, wide-bodied civil aircraft, about 25 years old which went out of service in the mid-1980s. A fleet of these old aircraft was purchased by the MoD for conversion into military tankers/troop carriers. A firm based at Cambridge was responsible for modifying the aircraft and supplying relevant data (e.g. installing new flight re-fuelling tanks, engine uprates and a rear cockpit/visual system). A US firm, Lockheed, the original aircraft producer, was to supply the all-important data package needed to drive the simulator.

Triumph, a one-off project, began in July 1994 and was due to be completed and in service by 9 September 1996 at the MoD. Engineers were drawn mostly from the military division of the company. The latter had been an independent firm recently acquired by Dynamics and amalgamated with Dynamics' civil operations. However, the contract had originated in the civil division and this led to continuity difficulties between the bid and early design stages. Another complication, discussed below, was that the two amalgamated companies were expected to operate under a new formal management system imposed by Dynamics.

3.2 Triumph project technology

The project cost was around £11 million which is average for a flight simulator. According to Dynamics engineers, Triumph was not unusual in terms of duration and risk, and there were no radical technical requirements which demanded new R&D. Around 20-25 engineers worked directly on Triumph at any one time. In addition, indirect company-funded managers and administrators from operations departments contributed to the project.

As with most flight simulator projects, Triumph had some special features which had to be accounted for. It was not typical of a military aircraft, because it was a conversion from civil to military, as noted. This meant that it had many 'civilian' aircraft features as well as military ones. The actual jet itself was a large civil one compared with the more usual, smaller, military jets.

The host computer and interfaces, based on a standard platform developed by the civil wing of the company, were to be tailored to meet Triumph's needs. The aircraft was fairly old and some of the original software was to be re-used. Around 50% of the flight simulator was new, including novel software which had to be designed using ADA. Although no novel technology had to be developed, some was new to the firm and various parts of the system were 'leading

---

scale complex projects. A review of DoD standards and their impact on industry is provided by Lake (1992), while Fairburn (1995) deals with multi-company projects. Although the approach has given rise to a huge literature (Chambers, 1986) it has little to say on how to deal with emergent, unexpected properties or A-type issues . A study by Walker et. al (1988) shows that large civil projects share many of the characteristics of military ones.

[9] ADA's development was sponsored by the US military (in the mid-1970s and early-1980s). It began to be used in flight simulation in specialist applications in the mid-1980s and became the dominant military language during the next ten years, displacing FORTRAN. ADA is a highly structured language developed for real time operations (as opposed to atemporal systems).

edge' and had to be configured in new ways. The most important technical challenge to the project team was the ADA language. This was the first time Dynamics had used ADA and this necessitated both formal training programmes and the learning of new skills and techniques on the part of several engineers.

3.3 Project organisation and key responsibilities

The Triumph project mapped on to the company (mostly engineering) organisational structure. The company had a typical project-based, matrix organisation with project teams cutting across functional lines. Functional departments supported the projects with services such as R&D, purchasing, marketing, finance, training and quality assurance.

The project team was led by a project manager (PM) and a systems engineer (SE). The PM, who's main responsibility was Triumph at the time, like most PMs was not a technically qualified engineer but carried out the business, financial, and day-to-day management of the project. The PM was in charge of overall milestones, plans and budgets and had to ensure schedules were kept, finances were in order and that team communications functioned well. He was the main interface with the client, dealing with customer problems, new requests and external communications. The SE, a senior engineer, was in charge of overall engineering, including systems integration of all the hardware and software. The SE, whose role was 'inward looking', was charged with ensuring all the engineering tasks were carried out according to specification, schedule and budget. The SE worked alongside the PM in what was by all accounts a good and effective working relationship.

It is worth noting that neither the PM nor the SE, as often occurs in matrix organisations, had a direct staff and worked mostly by facilitation and persuasion. Project team engineers reported up through functional lines to their respective Group Managers in charge of the four main engineering departments. However, considerable power of influence resided in the PM who reported regularly to company directors on progress and problems.

The engineering team leaders (TLs), in charge of work packages (further broken down into modules), negotiated with the PM and interpreted what the plans meant. The TLs then reported to their respective Group Managers to request resources (mostly people and machinery) to carry through fairly large work packages. TLs had between two and eight engineers reporting to them at any one point on Triumph. Some TLs had more responsibility than others. There were nine TLs in charge of avionics, tactics, radar, engines, controls, flight, systems integration, instructor operating system and computer systems.

In addition to the above, several senior engineering specialists had design responsibility for specific areas such as aircraft systems (including fuel, hydraulics and landing gear) and environmental control systems. Highly skilled and respected, they typically worked on several projects at a time, reporting both to TLs and Group Managers. Group Managers, the functional heads of the engineering departments, were responsible for resourcing Triumph and other projects. Group Managers, although highly influential, were not formally a part of the project structure. In Military there were four Group Managers responsible for the four main functional engineering areas.

3.4 Pressures on formal processes

At least three possible sources of strain on formal systems were evident in Triumph. Each could have potentially caused a deviation between A and B-type processes: (i) industrial turbulence; (ii) a new management system; and (iii) technical difficulties. First, general industrial turbulence and company transition affected Triumph as they did most other projects. For several years the flight simulator industry had been subjected to extreme distress, caused by market recession following the Gulf War and the decline in civilian aircraft travel, coupled with military cutbacks following the break-up of the Soviet Union. Recession had resulted in widespread lay-offs across the industry and total employment in Dynamics had fallen by more than 50% in the previous two to three years. Morale was low and job insecurity high. The bid phase of Triumph had been caught up in Dynamics' takeover of the civil company which entailed a physical site relocation. Constant change effected the company as did the loss of senior managers and key engineers, both before and during the Triumph project. In Triumph neither the PM nor the SE had been involved in the bid phase of the project due to the merger of the two companies which coincided with the early stage of the contract (see Part 4).

Second, the merger caused some confusion due to the coming together of two companies under a new management system, called MSA.[10] Following the takeover by Dynamics, the parent company began imposing MSA, a worldwide corporate system adapted to suit local circumstances. MSA covered company strategy and positioning, bidding procedures, project implementation, specific management tools, scheduling procedures, budgeting, cost control, risk management and purchasing. MSA was in the process of being introduced and some aspects had been adopted by Triumph.

Third, technical difficulties confronted the project. Triumph relied on a standard platform developed for civil flight simulators, but the engineers were drawn mostly from a military background, causing some difficulties at the early stage. Other technical difficulties, including unexpected requirements are analysed in Part 5. As discussed below, each of the above three factors played an important part in shaping the progress of the project.

PART 4: SOFTWARE TOOLS AND PROCEDURES

4.1 Formal tools and procedures

In theory, the formal software tools and procedures mandated by the company should have embodied and supported a B-type rational process. During our research we therefore identified 'hard' management tools which were used, and/or supposed to be used, by software engineers and managers.[11] Our aim was to assess how tools were used, if any tools were not used (and why) and what else, if anything, was being used to help get the job done.

---

[10] The name is changed to protect confidentiality. MSA was contained in a worldwide corporate toolbook, analysed during the research.

[11] For an analysis of different types of management tools, procedures and systems see Brady et al (1997).

We found it useful to divide management tools into three broad categories: (a) hard mandated tools, including computer-aided software engineering (CASE) tools, MoD requirements, PC-based tools and formal MSA reporting systems; (b) soft mandated tools such as software reviews (so-called code walkthroughs) and monthly coordination meetings to measure and communicate progress; and (c) soft informal tools (unofficial measures used on Triumph), including paper reports, *ad hoc* meetings and other forms of group interaction.

Many of the hard tools were defined by MSA which outlined procedures for bidding, project implementation, scheduling, software design and testing, budgeting, risk management and sub-contract management. Several hard tools, used in all UK military flight simulator projects, were contained in MoD standard 2167a. This called for Triumph to conform to a detailed design format to be sent to the MoD for approval. Similarly, formal software test procedures had to be followed and specific documents were to be used to record the project's history and track any changes. 2167A defined and documented the life cycle in a fairly rigid manner, with some tailoring permitted for flight simulation. 2167A provided a route map for Triumph, a work breakdown structure, high and low level design stages and documentation standards (called MoD JSP188 issue 4).

Other hard tools included ILS (Integrated Logistics Support standard SLIC-2B) adopted by the MoD from the US DoD (discussed below). Interleaf, a rigid documentation software package provided a template to ensure a common design format across the project. Matrix X V4, a rapid prototyping tool was to produce real-time ADA code of the model and then automate the documentation. In addition, there were many other hard tools and procedures in evidence in the large rule book for MSA.

4.2 How formal tools were used

As noted, we compared B-type tools with what actually took place in practice and identified problems in the use of tools. One finding was that some elements of MSA had been adopted by Triumph engineers (e.g. job titles and roles) and several engineers believed they were deploying MSA. However, it later emerged in discussions with directors that Triumph was not a company MSA project, a confusion which had resulted from the transition of one management system to another and somewhat unclear communications from top management.

Severe problems were reported with some IT-based tools (e.g. ILS). In theory, Triumph engineers should have fed into the ILS system all the items to be purchased, including data on repair advice and delivery information. The computer system would then place orders for the components just-in-time for their use. Triumph, like other projects, had to rely on ILS for the delivery of spares as the former manual system of order processing and chasing had been abandoned in favour of ILS. However, ILS was not capable of ensuring that some important spares (with lead times of eight to nine months) would be delivered on time. The just-in-time approach of ILS prevented ordering for inventory. In this case, based on his long experience, the PM by-passed the system unofficially (and against the company rules) to order spares directly (see Part 4). Similar events occurred in other projects in the company.

In Triumph the computer-based system, Compass Contract/MRP11, was (like ILS) according to engineers, not flexible enough for tailored, one-off projects. MRP11 was imposed on

project teams and in some cases engineers decided to by-pass the formal system to 'get the job done'. Interleaf, the documentation template, was generally viewed as useful, but engineers still had to draw many detailed arrows and lines by hand, because it was not flexible enough to encompass all the detail needed. The ways in which such lines were drawn differed from engineer to engineer.

One common complaint among engineers was that meeting the required standards entailed a great deal of time, effort and *post hoc* paperwork. Some felt too much 'engineering time' was spent on form-filling and bureaucracy which did not necessarily add value to the final product. However, the hard requirements mandated by the MoD were seen as part of the overall task funded by the customer, and were by and large planned for and implemented.

4.3 Informal soft tools and management discretion

The third category of tools, soft discretionary tools, were widely used. Indeed, the most important real triggers for action listed by Triumph engineers were informal tools, ranging from regular meetings to immediate responses to panic phone calls. One of the most cited and most useful tools was a two-page weekly written report by each engineer, introduced to Triumph by the PM. Although informal, it provided most of the basic data for managers and engineers to control and track Triumph's progress. By contrast the PM master schedule, which should have driven activities, was mostly used infrequently as a high level reference point by the PM.

Another important tool for communication and problem-solving was the weekly meetings which allowed regular interactions among individual software writers, different teams and managers. These meetings, which facilitated the exchange of data, the triggering of important actions and information feed-back from the customer, were viewed as critical to Triumph's progress. They followed an open format allowing conflicts to be identified and resolved. The weekly meetings, which were optional from the PM's perspective, did not take place in all projects in Dynamics and were not mandated.

Evidence of discretion and human initiative abounded in Triumph and in other company projects. In Triumph, some of the most important project management features resulted from personalities rather than formal tools and systems. Regardless of official procedures, the SE and PM had substantial degrees of freedom in the styles and methods by which they pursued their goals. Both initiated new procedures and chose how to deal with risk, communications and team motivation in a manner which suited their team, their past experiences and what their immediate bosses expected of them.

At the systems integration level, the Triumph SE believed that the best way to meet both software and hardware goals was to work closely with the engineers on a day-to-day basis, leaving customer interfacing (and other outside business tasks) to the PM.[12] This division of work appeared to work well, according to interviewees.

---

[12] In much larger projects (e.g. those with hundreds of engineers) more formal approaches usually have to be adopted. For an account of how to scale up effective informal systems while maintaining the benefits of smaller groups see Culver-Lozo (1995).

At the software practitioner level, TLs, engineers and specialists exercised a great deal of discretion as to how to implement software tasks and how to approach the detailed design and testing of modules. Practitioners had complex choices to make and were responsible for managing not only their own work packages, but also the interface between their work and that of other engineers. To some extent, software engineers were also managers, selecting, choosing and adapting management tools and documentation in order to complete their tasks effectively.

In short, Triumph relied heavily on personalities and discretionary choices. While high level formal procedures established a broad framework for operating, as well as deadlines and milestones, personal qualities (especially inter-personal skills) were also important. In Triumph, as in other projects, the tacit skills of the PM, the SE, and other team members played an important part in the project's progress and problems. To understand why unofficial soft tools appeared so important to the project it is helpful to turn to actual software work processes in Dynamics, and to compare these with rational ones.

PART 5: A-TYPE ACTUAL *VS* B-TYPE RATIONAL PROCESSES

5.1 Seven major deviations between rational and actual

Our research revealed seven major deviations between A- and B-type processes, most of which created uncertainty and caused risks to the project. Here we briefly describe the rational process as contained in official company manuals and procedures, how the actual processes differed, the causes of differences, the outcomes and how risks were being managed in Dynamics. The seven major deviations in the software process are presented in Table 2 and commented on below, in turn. Many of these problems were general difficulties, common both to Triumph and to other projects at Dynamics.

Table 2: Deviations Between Rational and Actual Processes*

1. Overall project management processes
2. The requirements capture process
3. The detailed design process
4. Testing procedures
5. Managing outside organisations
6. Measuring and monitoring of progress
7. Learning processes

* Each heading, apart from overall process (number 1), refers to project sub-processes. No order of priority is implied.

1. Overall project management processes

According to the formal system, Triumph software processes were to follow the overall requirements of MSA and 2167A, including a formal requirements analysis, monthly reports,

detailed design, off-line module testing and detailed documentation. A general high level representation of the software life cycle is presented in Figure 3.[13] The typical waterfall model includes requirements analysis, preliminary and detailed design, coding, testing, hardware-software integration and customer acceptance.

As shown below, what actually occurred was that tasks were not all completed according to schedule. Nevertheless, other parts of the project moved forward leading to variations in progress within the project. Incomplete information, and unexpected feedback loops from later to earlier stages necessitated major revisions to plan. Not all changes were properly documented and new requirements emerged which could not been foreseen.

One main reason why actual events proceeded in a more haphazard way than the rational process was the intense work pressures placed upon engineers and managers, because of market recession and redundancies. This context had, to some extent, depressed morale and introduced risks to both schedule and to technical targets. Software testing was rushed and informal as insufficient time was available for code walkthroughs. Cross programme scheduling loops occurred as key engineers were pulled out of Triumph to engage in 'bursts' of problem solving in other projects.

Other reasons for variation between A and B processes included organisational tensions inherent in the matrix structure. In Triumph, as in other projects, the PM and SE had high levels of project responsibility, but no direct staff. They relied on Group Managers for key people and had to compete for resources with five or six other large military projects underway in the company - 'too much responsibility but too little power' as one put it.

In response, the PM and SE fell back on soft, informal tools to track events, control budgets, ensure progress and deal with unexpected risks. Most problem-solving was conducted amicably through informal discussions with TLs and their Group Managers. Within engineering, there existed a culture of positive problem-solving, fairly typical in engineering environments. Because, the PM reported monthly to a group of company directors, this enhanced his status in relation to Group Managers and provided a final source of backup in case of severe problems or conflicts.

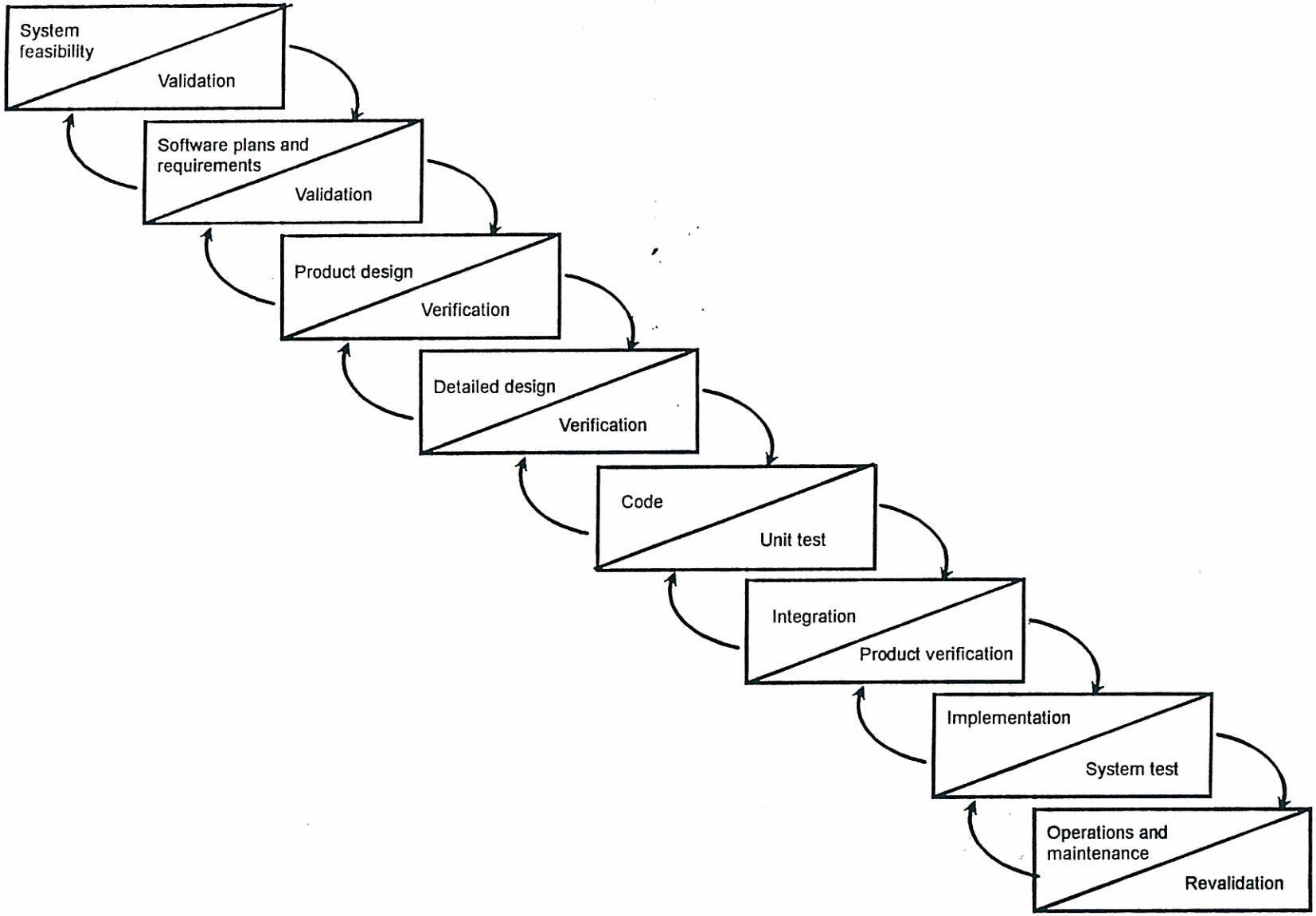
2. The requirements capture process

In theory, according to company procedures a major engineering effort should have gone into the requirements capture process to ensure all major customer needs were costed, understood and planned for. In practice, too little engineering input took place at the bid stage. Promises were made which were difficult to fulfil and not fully budgeted for. The reason for this was that due to stiff competition company negotiators believed they had to grant concessions to the customer at short notice to secure the order (a common problem in flight simulator bids).

---

[13] Also see Table 1 for a detailed breakdown of project stages and timing. The company had its own proprietary, detailed flow chart which cannot be shown here. However, Boehm's standard waterfall model is close enough for the present purposes.

Figure 3: The Waterfall Model of the Software Life Cycle



Source: Boehm (1983) p7

In addition, some early requirements which were captured were not communicated sufficiently well from the bid team to the project team. Because of the company takeover, neither the PM nor the SE were involved in the bid phase causing problems in both technical and managerial continuity between the bid and requirements analysis stages. Staff shortages, the company merger and site relocation meant that delays and uncertainty had occurred for three or four months and the requirements analysis phase had been rushed.

Later on, other new requirements emerged which could not have been captured at the early design stage. Such emergent properties often complicate the rational project cycle in flight simulation. For example, in Triumph, a malfunction in the rear visual system was only picked up by a pilot at a late stage and this called for revisions to the overall system design. The risks to the project were being managed through close cooperation between engineers and the customer, and the help of specialist engineers, able to ensure that revisions to other design modules could be kept to a minimum.

3. The detailed design process

According to the B-type rational process, once the requirements were captured and broken down into modules, engineers should then move to the detailed design stage. However, some software engineers jumped straight from the requirements analysis and preliminary design stages to software coding. This occurred not only in Triumph but in other projects as well. Pressures of work caused some to skip the formal design steps. Some experienced engineers felt they knew how to produce the code directly and did not need to conduct detailed design. Others were not sufficiently aware of detailed design procedures. In general, too few engineers used (e.g. CASE) tools to do detailed design. Practices varied across departments and projects. One of the risks caused by the lack of attention to detailed design was that the number of coding errors could increase and that these might not be picked up until a later stage where corrections would be difficult and costly. A similar problem appeared in testing.

4. Testing processes

In theory, all testing should have followed requirements, formal test procedures and code walkthroughs of software modules.[14] However, most testing was informal self-test, where engineers typically set up and ran a few testing parameters. A large amount of testing went on during later stages (e.g. hardware/software integration) when correction costs were high and technically difficult to conduct. Existing hard procedures (e.g. the customer Acceptance Test Schedule) were too general to ensure the testing of individual software components. Managers were attempting to resolve the detailed design and testing problems by defining the processes for both more clearly in Dynamics. Requirements for the testing of modules were being made more transparent and, in some cases, more stringent. Also, simple metrics for tracking and measuring the impact of good test practices were being developed.

---

[14] See Yourdon (1978) for an analysis of the benefits of code walkthroughs.

5. *Managing outside organisations*

The process of managing outside suppliers of software and data fell into two categories: (a) customer interactions and (b) sub-contractor and supplier management.

(a) Customer Interactions
Although the formal system made allowances for customer interactions in the design and development process at the early phase, most engineers believed that customer feedback was central throughout the project life cycle and not just at the early requirements capture stage. As Triumph progressed, new needs emerged and client requirements were changed. Software practitioners had to respond to these changes and this complicated the rational software life cycle. However, in Triumph as in other projects, there was insufficient customer interfacing planned informally at the engineering level, as requirements were supposed to be captured early on. Engineers responded to difficulties as they arose, used their experience and judgement to pre-empt problems and employed their discretion on deciding on particular courses of action to resolve problems.

(b) Sub-contractors and suppliers
Along with customers, major suppliers were on the critical path of software development and systems integration, a feature not recognised in the formal company process chart, nor the simple model of Figure 3 above. In Triumph, as in most projects, some difficulties (e.g. delays and technical problems) resulted from difficulties with suppliers. Engineers responded by working more closely and regularly on a co-engineering basis with suppliers to help identify and manage supplier risks.

Such difficulties were compounded in the case of Triumph by the rigidity of the formal ILS system for purchasing. This rigidity convinced the PM that the best way to ensure the necessary spare parts were obtained from suppliers on time was to by-pass the formal system. Managers felt forced to take unsanctioned actions to mitigate the risks introduced by ILS.

6. *Measuring and monitoring of progress*

A variety of data collection and metrics tasks were called for in the formal company manual. However, little data was actually collected on software performance and productivity. The data which were available could not be used to track and improve performance in software. In addition, the accounting practice in Dynamics of booking general infrastructural work to specific projects tended to mask true project costs. Although engineers recognised that simple, clear and useful metrics were needed to track and improve software performance, these were not available, nor was there any agreement on what form such metrics should take and how they could be developed and used.

7. *Learning processes*

One very important task in project-based organisations is to capture and transfer knowledge from one project to another, thereby increasing learning, ameliorating risks and raising

productivity.[15] The formal system involved a post-project review contained in the company toolkit. While Triumph had a brief review, in many other cases post-project meetings were not conducted. Overall, there was no basic system in place for analysing and reflecting on projects and feeding back key lessons (positive and negative) into the organisation. Constraints on time and resources prevented more practitioners from engaging in post-project learning assessments. Knowledge sharing was largely informal as individuals gained experience and moved from one project to another.

A 'market' for experienced people had developed with capable, knowledgeable and experienced individuals in high demand from competing PMs and departmental heads. Because learning was haphazard and focused on individuals, there was a risk that good practices and successes might not be repeatable.

5.2 Is the Triumph case generalisable?

The above evidence indicates considerable deviation between A- and B-type processes in Triumph. It should be noted that Dynamics is a highly respected, successful world leader in flight simulators for both the civil and military fields. As such, it is reasonable to ask: (1) why did actual processes differ so much from rational ones? and (2) are the underlying causes particular to Dynamics or flight simulation or can they be expected to occur in other products, companies or sectors?

From the Triumph case we can identify at least seven causes of divergence between A- and B-type processes: first, the turbulent, high pressured business environment; second, the complexity of the product and project tasks; third, difficulties in precisely capturing customer requirements at the start of, and during, the project; fourth, the need to progress with incomplete information due to feedback loops between later and earlier stages of production and (fifth), new unpredictable requirements emerging during production; sixth, heavy work pressures among staff; and seventh, the role of outside organisations with conflicting business goals and motivations.

Regarding the high pressured work environment, itself partly caused by industrial turbulence, today, such conditions are by no means uncommon. Indeed, fast changing markets face many firms, even in sectors once noted for their stability (e.g. aerospace, the utilities, military and shipbuilding). Other causes of work pressure included employee redundancies ('downsizing') and new management processes, again not uncommon, especially in high technology fields.

Regarding product and task complexity, although in many industries products are much simpler than in flight simulation, in others they are equally or more complex, requiring multiple sets of components, skills and knowledge. For example, hospital information support systems, air traffic control systems, avionics systems, intelligent buildings, oil drilling equipment and aircraft engines. As with simulators, these are complex systemic products involving an important component of embedded software. Most are project-based and the user is often involved in design, installation, maintenance and post-delivery modifications. While issues of

---

[15] The importance of organisational learning to productivity and competitiveness is now well-recognised (Senge, 1990; Leonard-Barton, 1988; Garvin, 1993; Stata, 1989; Malerba, 1992).

this kind may not apply to low cost, mass produced goods based on standard components (e.g. cars, TVs and camcorders) it is likely to apply to other CoPS areas to some extent.[16]

Uncertainty, incomplete information, difficulties in fully capturing user requirements and the emergence of new unpredictable system requirements during production are all closely related. Again, such properties are to be expected in CoPS with multiple interfaces and varied knowledge bases (Boardman, 1990), although they probably do not apply to low cost, simpler products.

The failure of suppliers to deliver key inputs is fairly common in major projects which involve many firms, but probably less common in projects carried out largely in-house. The trend in some CoPS industries for major producers to outsource production and become systems integrators (e.g. in aeroengine and warship production) has occurred in response to increasing product complexity, the rise of innovative new input manufacturers from, for example, the semiconductor and computer industries (Miller et al. 1995) and the strategic desire to focus on core competencies by restricting the range of in-house component production activities.

Outside organisations may not share the same project or business goals of the prime contractor and may have different company cultures and ways of working. It may not always be possible (or even desirable) to attempt to force outside organisations to comply strictly with B-type formal in-company processes. In major projects, A-type soft factors such as trust, goodwill, negotiation skills and persuasion are likely to become very important to project progress and success.

When all or most of the above causal factors apply, then deviations between A- and B-type processes of the kind witnessed in Triumph may be replicated in other sectors, as might the outcomes in terms of project risks and difficulties. If the above factors apply with less force (e.g. because production occurs largely within one firm or because there is industrial stability and little work pressure) then the deviation may be expected to be less intense, and actual processes may approximate the rational more closely. Conversely, large CoPS projects with very high levels of technical complexity and novelty and many sub-contractors might expect to find more severe divergencies between A- and B-type processes and more intense problems in completing projects on time and within budget.

In circumstances where the above factors apply then, as in the Triumph case, it is unlikely that B-type systems alone will be adequate to the task at hand. The need for speed, flexibility, human initiative, rapid problem solving and adjustment to new technical circumstances will place great pressures on the rational B-type systems. In response to uncertainty, engineers are likely to fall back on A-type soft systems to enable them to resolve and avoid problems, mitigate risks and possibly even to turn circumstances to their advantage.[17] Experienced engineers will not fully trust elaborate rational systems, particularly if they are imposed from above and will often use experience and judgement to make discretionary choices. Such

---

[16] See Hobday (1998) for an analysis of the critical product features of CoPS, along with many other examples.

[17] For example, in one software module in Triumph person-hours had been over-estimated leading to a budget surplus in this particular area.

actions do not necessarily reflect a failure to follow company rules, but rather a failure of the procedures themselves and, in the case of Triumph, a strong desire to meet the customer's requirements and secure follow on orders.

PART 6: REPRESENTING A-TYPE SOFTWARE PROCESSES

6.1 A partial representation of process stages

To highlight typical problems faced in managing complex software processes, this section attempts to outline the interdependencies and feedback loops found in Triumph and possibly in other complex, multi-firm projects. The model does not attempt to illustrate all informal A-type processes, but focuses on process stages to illustrate the progressive narrowing of options and reduction of risk/uncertainty found in engineering projects involving a major software component.[18]

Figure 3's waterfall model of the software process failed to illustrate the overlap between stages and the feedback loops between users, suppliers and other projects.[19] In practice, key engineers were pulled out of their allocated projects to solve problems on other projects, causing cross-project scheduling loops, technical problems and knock on delays. Figure 4 provides a rough approximation of actual simulator software progress, to be contrasted with the neat formal process of Figure 3. Figure 4 is developed in four stages to show how the layers of process complexity build upon each other, resulting in a complicated set of feedback loops between stages, with suppliers and customers and other projects proceeding simultaneously in the organisation.[20]

Figure 4.1 shows the main stages (a) and their overlaps (b). This corresponds to Figure 3, but highlights the overlap between stages both in terms of time and technical feedback. Often, for example, the detailed design will begin before the preliminary design is finished and will feed back valuable information to the preliminary design. The spheres, which represent each main activity, become smaller and smaller as the final stage of customer acceptance is approached. The size of the sphere indicates the degree of imprecision and uncertainty (and risk) in relation to the final output. As the project proceeds uncertainty and risk diminish, options are narrowed, learning occurs until the simulator is completed successfully.

Figure 4.2 includes (a) and (b) and adds feedback loops (c) from later to earlier stages. In simulators new unforeseen requirements can emerge fairly late on in the project cycle (e.g. from input from the human pilot), which requires revisions to the preliminary design or

---

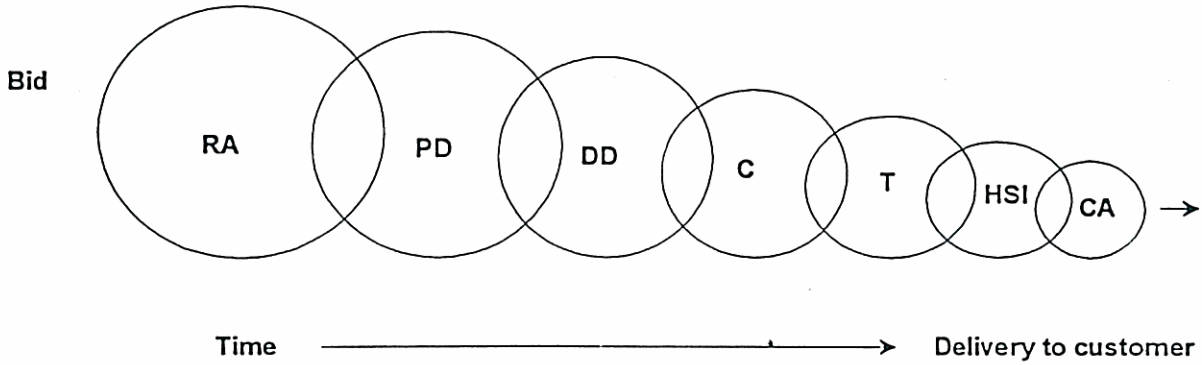
[18] The idea that projects follow a process of option/uncertainty narrowing through learning was first put forward by authors such as Lindblom (1959) and Klein (1962).

[19] More sophisticated models such as the spiral model also tend to ignore overlaps, inter-company feedback loops and informal processes (Boehm, 1988). An excellent exception is also provided by Boehm (1989a), who addresses the need for negotiation skills in multi-company software projects.

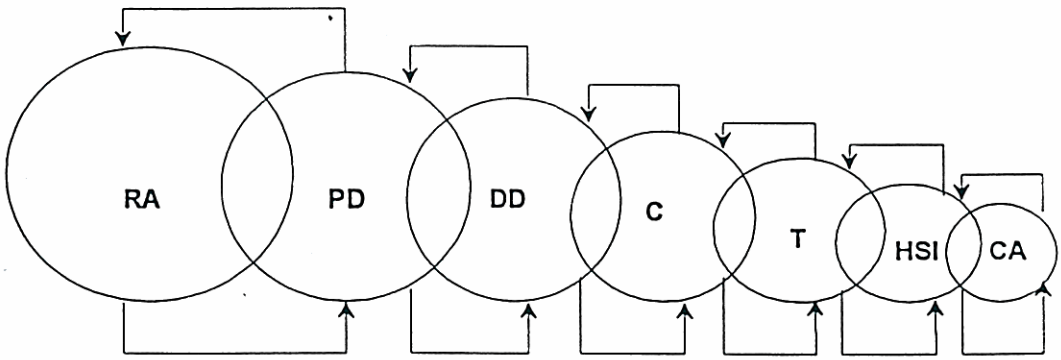
[20] To keep the representation manageable we omit other important connections, notably links with the functional departments which often support projects.

Figure 4: An Approximation of Actual Software Processes, Including Option Narrowing and Feedback Loops

4.1 Main stages, (a) including overlaps (b)



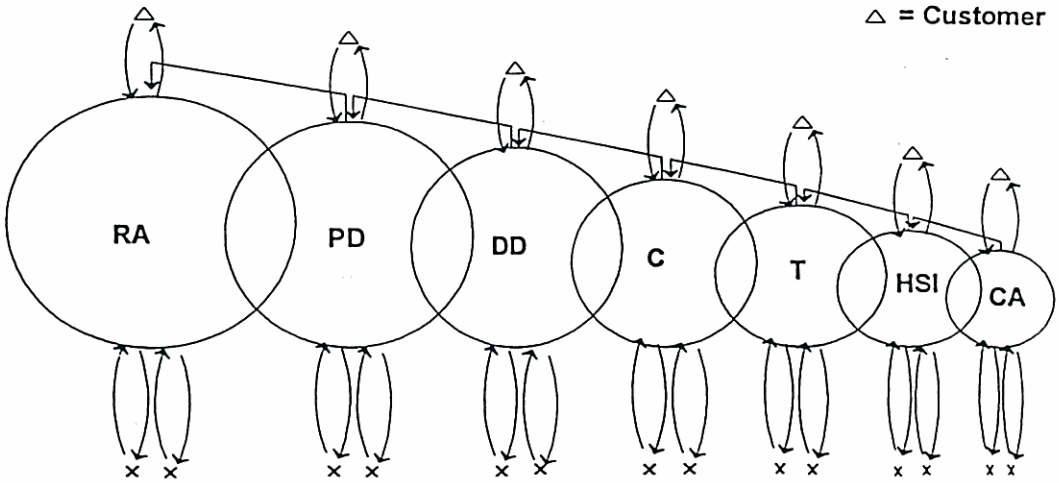
4.2 Main stages (a) overlaps (b) and feedback loops, from latter to earlier stages



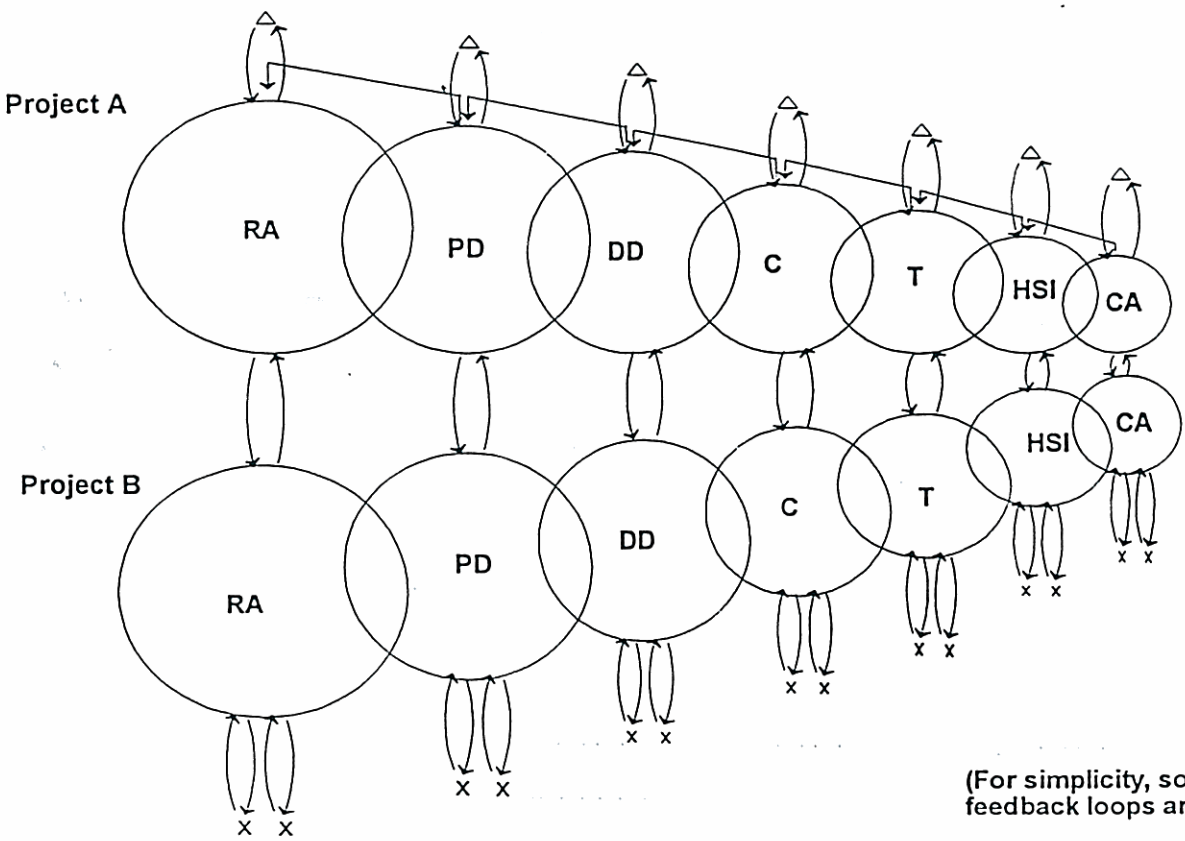
Codes:

RA	= Requirement Analysis
PD	= Preliminary Design
DD	= Detailed Design
C	= Coding
T	= Testing
HSI	= Hardware-Software Integration
CA	= Customer Acceptance

4.3 (a) (b) (c) plus customer feedback loops (d) and key supplier feedback loops (e)



4.4 (a) to (e) plus feedback loops from other projects (f)



(For simplicity, some feedback loops are omitted)

indeed the requirements analysis. Such changes, if major, can have system wide knock on effects and require changes in several software modules and their interfaces.

Figure 4.3 adds customer feedback loops (d) and key supplier feedback loops (e). The customer, represented by a triangle, is able to feed in at all stages of the work, and especially at the early and late stages. During the requirements analysis, very close customer interaction is expected. During testing, often the client will wish to see, approve and comment on test procedures and schedules. In the case of civil simulators the regulator may also play a part during these activities. Equally, the key suppliers represented by 'X' are critical to the project's progress. In the case of Triumph, Lockheed supplied the data package, Evans and Sutherland the main visual system, Concurrent Computer Corporation the computer systems and training in the ADA language. Thus, the process is one of co-engineering as the partners work with each other, share data and feed the systems integrator with key components.

Finally, Figure 4.4 draws back to show how one project also has feedback loops with other projects (f) proceeding at the same time in the organisation. In some cases, key engineers are withdrawn from projects (as in Triumph) to work on other projects. Equipment and other resources have to be shared. Despite the efforts of PMs to retain a coherent project team, emergencies arise on other projects and new bids occur which draw resources and priority away from the project in question. In fact, Figure 4.4 understates the complexity of cross-project scheduling loops. There may be several projects which relate to each other in a variety of ways. Some of the engineers on Triumph, for example, worked on four or five projects at any one time, although these individuals were exceptions.

6.2 Possible benefits from a representation of actual processes

Viewing the project in the light of Figure 4.4 points to the causes and some of the implications of actual software processes at work, although it does not capture the A-type processes themselves. Constant human actions and reactions have to take place to deal with uncertainty, emergent properties and multiple feedback loops. Figure 4.4 suggests the importance of informal communications, meetings, workshops, motivation and good team relations for the effective completion of complex projects. Without these, despite highly elaborate B-type procedures, the project would stand little chance of successful completion.

The representation in Figure 4.4 also points to potential areas of difficulty for those involved in flight simulator projects and perhaps other projects facing the conditions outlined in Section 5.3. The real process model implies that informality is likely to come into play and that healthy soft processes such as team building, communications, decision-taking under pressure and uncertainty, time management and human motivation are essential to project efficiency and effectiveness. The model could potentially help to alert senior managers and software analysts to some of the real problems facing project teams and help initiate new engineers into the complicated real world of software engineering.

CONCLUSION: IS THE RATIONAL APPROACH IRRATIONAL IN COPS?

The study explored the variations B-type (rational) and A-type (real) software processes in a complex product, arguing that these divergencies have significant implications for management theory and practice. The paper identified at least seven causes of variation, many of which contributed to uncertainty in estimation, decision making, project management and project outcomes: first, strains on formal procedures brought about by a turbulent business environment; second, the difficulty and complexity of the project tasks; third, difficulties in precisely capturing customer requirements at the start of, and during, the project; fourth, the need to progress from one stage to another with incomplete information due to feedback loops between later and earlier stages of production; fifth, the emergence of new unpredictable requirements during production; sixth, heavy, often distracting, work pressures; and seventh, the involvement of outside organisations with conflicting business goals, motivations and company cultures on the critical path of project completion. Under these circumstances, B-type systems were inadequate and managers and practitioners fell back on informal soft A-type practices. A-type real practices were often masked and hidden from upper management. However, although they ran contrary to official B-type processes they enabled engineers to 'get the job done'.

Although one cannot generalise from a single case, the paper proposed that in other CoPS where project decisions are taken under conditions of uncertainty, then engineers are also likely to rely to some extent on soft informal systems. Under these conditions, A-type human initiatives, tacit knowledge, soft tools and fast informal communications become essential to task completion. In the case of Triumph, soft tools were used to trigger activities, monitor progress, resolve problems rapidly, deal with customer difficulties and manage risks caused by unexpected real world problems. Experienced engineers used their discretion, experience and judgement to respond to the uncertain, complicated and changing environment. In other CoPS cases where tasks are complex and customers and suppliers are directly involved in project design and execution, then one might expect to see significant divergencies between B- and A-type processes.

If conflict between A and B is often the case in CoPS, then the question arises - is it irrational to proceed with B-type tools and procedures if they are detached from reality and unable to cope? To answer this question, it is necessary to assess both the benefits and disadvantages of the rational approach. Contrary to the assumptions of some organisation analysts (Section 1.2), B-type processes have important benefits. In the case study, B-type systems provided a useful (if inaccurate) route map to completion and a series of logical steps and targets to initiate actions (e.g. requirement capture and detailed design). They provided common software standards to aim for, both within the project and across the company. They also presented an ideal 'benchmark' for measuring progress, including a detailed timetable with milestones. Without an official, rational model of how to proceed, software engineering could descend into chaos with individuals acting independently of team goals. Formal procedures provided a single deterministic (if unrealistic) way forward where, in reality, there were perhaps dozens or hundreds of possible ways forward. Furthermore, the rational model provided some constraints on the creativity of individuals, and rules for the less creative to follow, both probably necessary for the successful management and completion of complex software projects.

However, the rational approach had severe disadvantages. It palpably failed to deal with many real world difficulties and realities and was therefore insufficient for the task at hand. It was silent on vital issues such as how to empower project managers, how to manage and motivate creative individuals, and how to capture and transfer learning across the organisation from exemplar projects. It had little or nothing to say on how to build effective project teams, how to communicate well, how to deal with and motivate suppliers, and how to understand and meet the changing and unpredictable needs of customers. Most importantly, the rational model could not cope with the reality of discretionary choice among software engineers. As the case showed, software engineers were not only practitioners responsible for carrying out prescribed tasks, but also managers (often creative ones) of complex work tasks and their interface with other professional groups. In the real world, it is both necessary and desirable that software engineers exercise a degree of discretion over procedures for task completion under conditions of task complexity, uncertainty and work pressure. Unlike standardised hardware manufacturing production, a large proportion of the software task is a highly skilled, design-intensive, individualistic activity which does not lend itself to a Taylorist, B-type routinised division of labour.[21]

In contrast to previous research on A *vs* B practices (Seely Brown and Duguid, 1991) these findings imply that it may well be *the manner in which B-type processes are created, valued, perceived and implemented*, which is the core of the A *vs* B problem, rather than the processes themselves. If B-type practices are developed and treated as 'canonical' rules which must be followed, and if they fail to accommodate the messy reality of much of the software task in CoPS, then they are likely to contribute to project failure rather than success, by masking real practices and widening the A *vs* B gap. Over time a widening of the A *vs* B gap is likely to lead not only to a lack of knowledge of real world problems on the part of senior management, but also imposition of unhelpful, time consuming bureaucratic procedures which hamper rather than assist real processes. Under such conditions, B-type organisational systems and structures are likely to give rise to control-centred interest groups whose existence may be threatened by the undermining of B-type authorities and by the empowerment and delegation essential for A-type realities. A large gap also implies a major difficulty for improving productivity and performance by learning from project-to-project as good innovative practice is driven underground and treated as illegitimate, finding itself in conflict with corporate culture and values. Nevertheless, much of the software field proceeds by ignoring Type-A realities and imposing more rigid B-type systems (e.g. Pulford et al, 1996; Jones, 1996) or by proceeding 'as if' A-type processes were rational (Parnas and Clements, 1986; Kellner, 1996).

Alternatively, it is possible that if B-type practices are treated as necessary but insufficient devices and designed to support real processes, then they are more likely to become valuable to and valued by practitioners. This, in turn, implies that practitioners must play a significant part in developing B-type procedures so that they become rooted in real practices, rather than B-type theories. If it is accepted by managers and practitioners that differences are likely to

---

[21] This may not be the case for less complex software tasks or projects involving mostly routine activities. Here Taylorism may be more applicable. Conversely, in the development phases of many new products and systems, uncertainty and learning are likely to heavily influence decision taking and other processes (Klein and Meckling, 1958).

exist between real and rational, but that B-type systems are needed to execute and manage projects effectively, then B-type procedures are likely to become more useful to practitioners, less bureaucratic, less controlling and more responsive to real A-type needs.

For many CoPS suppliers, this probably implies an ongoing effort to improve processes by identifying A *vs* B divergencies and narrowing the A *vs* B gap, which itself requires a senior management commitment to questioning official values, routines and organisational culture ('this is the way we do things around here'). By treating B-type processes as 'non-canonical' guides, it may well be possible to reduce the gap between management theory and actual practices, contributing to productivity and organisational effectiveness in CoPS.

REFERENCES

Abdel-Hamid, T. and Madnick, S. E. (1991): **Software Project Dynamics: an Integrated Approach**, Prentice Hall, New Jersey.

Baker, M. and Rouse, A. (1995): 'Getting and Keeping Software Quality Certification: some Associated Issues', in G. Doukidis, B. Galliers, T. Jelassi, H. Kremar and F. Land, (eds): **Proceedings of the 3rd European Conference on Information Systems**, Athens, Greece.

Barnard, C. I. (1938): **The Functions of the Executive**, Harvard University Press, Cambridge, Mass.

Boardman, J. (1990): **Systems Engineering: an Introduction**, Prentice Hall, New York.

Boehm, B. W. (1983): 'Seven Basic Principles of Software Engineering', **The Journal of Systems and Software**, Vol 3, pp3-24.

Boehm, B. W. (1988): 'A Spiral Model of Software Development and Enhancement', **IEEE Computer**, May, pp61-72.

Boehm, B. W. (1989): **Software Risk Management**, IEEE Computer Society Press, Washington, DC.

Boehm, B. W. (1989a): 'Theory-W Software Project Management: Principles and Examples, **IEEE Transactions on Software Engineering**, Vol. 15. No.7. July, pp902-916.

Boehm, B. W. (1991): 'Software Risk Management: Principles and Practices, **IEEE Software**, January, pp32-41.

Brady, T, Rush, H., Hobday, M., Davies, A, Probert, D. and Banerjee, S. (1997):'Tools for Technology Management: an Academic Perspective', **Technovation**, Volume 17, Number 8, 1997, pp417-425.

Burns, T. and Stalker, G. M. (1961): **The Management of Innovation**, Tavistock, London.

Chambers, G.L. (1986): 'The Systems Engineering Process: A Technical Bibliography, **IEEE Transactions on Systems, Man and Cybernetics**, Volume SMC-16, No.5, September/October, pp712-722.

Collins, T. and Bicknell, D: (1997): **Crash**, Simon & Schuster, London.

Culver-Lozo, K. (1995): 'Software Process Iteration on Large Projects: Challenges, Strategies and Experiences', **Software Process: Improvement and Practice**, Volume 1, pp35-45.

Davenport, T.H. (1993): **Process Innovation: Reengineering Work through Information Technology**, Harvard Business School, Boston, Mass.

Davenport, T.H. (1996): 'Why Engineering Failed: the Fad that Forgot People', **Fast Company**, Premiere Issue pp70-74, Boston, Mass.

DeMarco, T. (1979): **Concise Notes on Software Engineering**, Yourdon Press, New York.

DeMarco, T. and Miller, A: (1996): 'Managing Large Software Projects', **IEEE Software**, July, pp24-27.

Drucker, P. F. (1977): **Management**, Pan Business Management.

Emam, E. El and Madhavji, N. H. (1995): 'The Reliability of Measuring Organisational Maturity', **Software Process: Improvement and Practice**, Volume 1, pp3-25.

Fairburn, A.G. (1995): **A Systems Approach to Ventures**, IEE Review, September, pp195-198.

Follett, M. P. (1918): **The New State**, Longmans, London.

Garvin, D. A. (1993): Building a Learning Organization, **Harvard Business Review**, July-August, pp78-92.

Gibbs, W. W. (1994): 'Software's Chronic Crisis', **Scientific American**, September, pp72-81.

Graham, P. (1995): **Mary Parker Follett - Prophet of Management**, Harvard Business School Press, Boston Massachusetts.

Hamel, G. and Prahalad, C. K. (1994): **Competing For the Future: Breakthrough Strategies for Seizing Control of Industry and Creating Markets of Tomorrow**, Harvard Business School Press, Boston MA.

Hammer, M. and Champy, J. (1993): **Reengineering the Corporation: a Manifesto for Business Revolution**, Nicholas Brealey, London.

Hilmer, F. G. and Donaldson, L. (1996): **Management Redeemed: Debunking the Fads that Undermine our Corporations**, The Free Press, New York.

Hobday, M. G. (1998): 'Product Complexity, Innovation and Industrial Organisation', **Research Policy**, Vol. 26, pp688-710.

Hobday, M. G. and Brady, T. (1997): A Fast Method for Analysing and Improving Complex Software Processes', SPRU/CENTRIM, CoPS Working Paper, Submitted to **R&D Management**.

Huber, G. P. (1996): 'Organizational Learning: the Contributing Processes and the Literatures, Chapter 6, in M. D. Cohen and L. S. Sproull (eds), **Organizational Learning**, Sage Publication, California.

Humphrey, W.S. (1989): **A Discipline for Software Engineering**, Addison-Wesley, Reading, Mass.

Humphrey, W.S. (1995): **Managing the Software Process**, Addison-Wesley, Reading, Mass.

Jones, C. (1996): 'Our Worst Current Development Practices', **IEEE Software**, March, pp102-104.

Kellner, M. I. (1996): **Business Process Modeling: Lessons and Tools from the Software World**, Software Engineering Institute, Carnegie Mellon University.

Klein, B. H. (1962): 'The Decision Making Problem in Development', in **The Rate and Direction of Inventive Activity: Economic and Social Factors**, Conference of the Universities-National Bureau Committee for Economic Research and the Committee of the Social Science Research Council, Princeton University Press, Princeton.

Klein, B. and Meckling, W. (1958): 'Application of Operations Research to Development Decisions', **Operations Research**, May-June, pp352-363.

Lake, J. G. (1992): 'Systems Engineering Re-Energized: Impacts of the Revised DoD Acquisition Process', **Engineering Management Journal**, Volume 4, No. 3, September, pp8-14.

Leonard-Barton, D. (1988): 'Implementation as Mutual Adaptation of Technology and Organization', **Research Policy**, Vol. 17, pp251-267.

Lindblom (1959): 'The Science of "Muddling Through" ', **Public Administration Review,** American Society for Public Administration, Washington DC, Vol. 19. Spring, pp.78-88.

Littlewood, B. and Strigini, L. (1992): 'The Risks of Software', **Scientific American**, November, pp38-43.

Marschak, T. A. (1962): 'Strategy and Organization in a System Development Project', in **The Rate and Direction of Inventive Activity: Economic and Social Factors**, Conference of the Universities-National Bureau Committee for Economic Research and the Committee of the Social Science Research Council, Princeton University Press, Princeton.

Malerba, F. (1992), 'Learning by Firms and Incremental Technical Change', **The Economic Journal**, Vol. 102, July, pp.845-859.

Miller, R. , Hobday, M., Leroux-Demers and Olleros, X. (1995): 'Innovation in Complex Systems Industries: the Case of Flight Simulation', **Industrial and Corporate Change**, Vol. 4, Number 2, pp363-400.

Mintzberg, H. (1979): **The Structuring of Organizations**, Englewood Cliffs, New Jersey.

Mintzberg, H. (1989): **Mintzberg on Management: Inside Our Strange World of Organizations**, The Free Press, New York.

Morgan, G. (1986): **Images of Organization**, Sage, London

Orr, J. (1987): 'Narratives at Work: Story Telling as Cooperative Diagnostic Activity', **Field Service Manager**, June, pp47-60.

Orr, J. (1990): 'Talking About Machines: an Ethnography of a Modern Job', Doctoral Dissertation, Cornell University.

Parnas, D.L. and Clements, P. C. (1986): 'A Rational Design Process: How and Why to Fake It', **IEEE Transactions on Software Engineering**, Vol. SE-12, February, pp251-257.

Paulk, M.C. (1993): 'Capability Maturity Model, Version 1.1', **IEEE Software**, Vol. 10, No.4, pp18-27.

Paulk, M. C. (1995): 'The Evolution of the SEI's Capability Maturity Model for Software', **Software Process: Improvement and Practice**, Pilot Issue, pp3-15.

Paulk, M. C. (1995a): 'The Rational Planning of (Software) Projects', **Proceedings of the First World Congress for Software Quality**, San Francisco, CA 20-22 June, Section 4.

Pulford, K., Kuntzmann-Combelles, A., Shirlaw, S. (1996): **A Quantitative Approach to Software Management**, Addison Wesley, Wokingham, England.

Quinn, J. B. (1980): **Strategies for Change: Logical Inrementalism**, Irwin, Homewood, Ill.

Quintas, P. (1995): **Software Innovation in the Context of Complex Product Systems**, Draft Working Paper, CENTRIM-SPRU-OU Complex Product Systems Project, EPSRC Technology Management Initiative.

Ritchie, B. and Marshall, D. (1993): **Business Risk Management**, Chapman and Hall, London.

Rout, T. P. (1995): 'SPICE: A Framework for Software Process Assessment', **Software Process: Improvement and Practice**, Pilot Issue, pp57-66.

Sage, A.P. (1981): 'Systems Engineering: Fundamental Limits and Future Prospects', **Proceedings of the IEEE**, Volume 69, No. 2, February, pp158-166.

Sapolsky, H. M. (1972): **The Polaris System Development: Bureaucratic and Programmatic Success in Government**, Harvard University Press, Mass.

Seeley Brown, J. and Duguid, P. (1996): 'Organizational Learning and Communities of Practice: Towards a Unified View of Working, Learning, and Innovation', Chapter 3, in M. D. Cohen and L. S. Sproull (eds), **Organizational Learning**, Sage Publication, California.

Senge, P. M. (1990): The Leader's New Work: Building Learning Organizations, **Sloan Management Review**, Number 7, Fall, pp7-23.

Shenhar, A. (1994): 'Systems Engineering Management: A Framework for the Development of a Multidisciplinary Discipline', **IEEE Transactions on Systems, Man, and Cybernetics**, Volume 24, No.2, February, pp327-332.

Shtub, A. Bard, J.F. and Globerson, S. (1994): **Project Management: Engineering, Technology and Implementation**, Prentice Hall, New Jersey.

Simon, H.A (1955): 'A Behavioural Model of Rational Choice', **Quarterly Journal of Economics**, 69, 99-118.

Smith, D.J. (1994): 'Software Quality and Reliability', D. Lock (ed.): **Gower Handbook of Quality Management**, Gower, Hampshire, England, Second Ed.

Stata, R. (1989): 'Organisational Learning - the Key to Management Innovation', **Sloan Management Review**, Number 63, Spring, pp63-74.

Taylor, F.W. **Principles and Methods of Scientific Management**, Harper and Row, New York, 1911.

Walker, W., Graham, M. and Harbor, B. (1988): 'From Components to Integrated Systems: Technological Diversity and Integration between the Military and Civilian Sectors', in P. Gummett and J. Reppy (eds.), **The Relations Between Defence and Civil Technologies**, Kluwer Academic Publishers, London.

Woodward, J. (1958): **Management and Technology**, London, Her Majesty's Stationary Office, London.

Webb, A. (1994): **Managing Innovative Projects**, Chapman and Hall, London.

Yourdon, E. (1978): **Structured Walkthroughs**, Yourdin Inc., New York, Second Ed.

Annex 1: Capability Maturity Model (CMM)

CMM is based on work by the Software Engineering Institute (SEI) of Carnegie Mellon University (Pittsburgh, Pennsylvania). The basic idea is that the quality of software output is determined by the quality of the organisation's software processes. CMM is based on the application of Shewart/Deming's statistical process control (SPC) and total quality management (TQM) techniques to software. As a company's capability matures, software becomes better defined and more consistently implemented throughout the organisation. The promise of CMM is improved productivity, quality, predictability and reputation. CMM is now a fully-defined model, backed by the US Department of Defense.

However, CMM is not a guarantied solution. Good quality software will still depend on underlying organisational factors (e.g. motivation, learning and communications). Also, CMM is not yet domain specific. It is still evolving and may need tailoring for individual sectors such as telecommunications and flight simulation. The five basic CMM software process levels are described below.

CMM Software Process - Levels 1 to 5

- Level 1 Initial - ad hoc/ chaotic - few processes defined for developing and maintaining software - success depends on key individuals
- Level 2 Repeatable - basic project management techniques used to track cost, schedule and functionality - previous (similar) project successes can be repeated
- Level 3 Defined - documented processes for both management and engineers - accepted standard software practices across all projects - tailored versions possible
- Level 4 Managed - detailed measures of both software process and product quality collected, quantitatively understood and controlled
- Level 5 Optimising - quantitative feedback allows continuous process improvement from the process and from innovative ideas and technologies

Levels 1 and 2 focus on project management. Levels 3 to 5 focus on understanding and improving the process with feedback data.