# SOFTWARE ENGINEERING PRACTICES AND TOOL SUPPORT: AN EXPLORATORY STUDY IN NEW ZEALAND

Elizabeth A. Kemp, Chris Phillips, Jaimee Alam,
Institute of Information Sciences and Technology
College of Sciences
Massey University
Palmerston North
Email: e.kemp@massey.ac.nz, c.phillips@massey.ac.nz, jaimee_2@hotmail.com

## ABSTRACT

This study was designed as a preliminary investigation of the practices of software engineers within New Zealand, including their use of development tools. The project involved a review of relevant literature on software engineering and CASE tools, the development and testing of an interview protocol, and structured interviews with five software engineers. This paper describes the project, presents the findings, examines the results in the context of the literature and outlines on-going funded work involving a larger survey.

***Keywords:*** *Project management, CASE tools, lifecycles, software engineers*

## INTRODUCTION

Research is currently being undertaken into the development of Computer Aided Software Engineering (CASE) tools to assist software engineers. A necessary first step is to find out about real users so that they can be provided with the kinds of tools that they find useful. There are few studies that attempt to model the activities of software engineers. Singer et al. (1997) commented on the paucity of studies in this area noting that little effort had been expended in understanding how software engineers work. Previous studies had focused solely on programmers who were usually university students working on small programs. They were themselves particularly interested in designing tools to help software engineers maintain legacy systems. Groves et al. (2000) have carried out a survey of software requirement specification practices in New Zealand but were primarily concerned with the requirements gathering activity. This study has a wider focus, with the project management procedures followed, the software process models adopted, and notations used all of interest. In addition, the CASE/I-CASE tools provided for staff and reasons for the use/non-use of such tools were also of concern. A preliminary study of the practices of software engineers in New Zealand, therefore, has been undertaken to inform debate in this area.

The project involved a review of relevant literature on software engineering and CASE tools, the development and testing of an interview protocol, and structured interviews with five software engineers. This paper describes the project, presents an analysis of the data, examines the results in the context of the theory described in the literature, and outlines on-going funded work involving a larger survey

## BACKGROUND

Software Engineering is a discipline involving the study of a systematic, quantifiable approach to the development, operation and maintenance of software (IEEE 1993). Software engineers can be considered as problem solvers (Pfleeger, 1998) aiming to produce quality software. Generally, software projects fall into the following categories: concept development, new application developments (e.g. meeting customer requests), application enhancements (corrections or upgrades), application maintenance or re-engineering legacy systems (Pressman, 2001). Another way of looking at a project is to determine what Groves et al. (2000) call the kind of development undertaken by a company. They distinguish between specific products for customers (one-off contracts or mass production), in house software to support the running of the organisation and product support where software is included in goods sold by the company. An organisation can be involved with more than one kind of development.

**Project Management**

It is necessary to have guidelines for the efficient development and evolution of software (Robillard et al., 2003). Typical project management activities include planning and scheduling, estimating costs, resource allocation, software configuration management, measurement, risk management, software quality assurance and formal technical reviews. There are no general sets of tasks that apply to every project. However, task sets (work tasks, milestones and deliverables) can be associated with various project types. The larger and more complex the application is, the more rigorous the set of management tasks should be (Wasserman, 1996). Three kinds of rigour can be defined (Presssman, 2001): strict, structured and casual. Strict refers to the situation where all lifecycle and project management activities are undertaken. Structured indicates that all the lifecycle and appropriate project management procedures necessary to produce high quality software are undertaken. Finally, casual does not imply that procedures are ignored but that all the lifecycle activities and a minimum set of project management tasks are undertaken. Various factors impact on the degree of rigour applied:

- Customer/developer communication
- Performance constraints
- Size of project
- Maturity of applicable technology
- Embedded or non-embedded characteristics
- Project staff

Rada and Craparo (2000) observe that the effective management of software engineering projects is an important issue for organisations. Whether or not the standards are developed internally, a company should follow any standard that it has selected.

It may be a mistake to assume that all software engineers follow the precepts of the many writers on software engineering. Behforooz and Hudson (1996) believe that software engineering practices and principles are not fully recognised by either academia or industry. Humphreys (1998) claims that the general practices of software engineers are poor by almost any measure. He substantiates this by describing the large number of reported systems (with defects) that are delivered late and over budget. He may well be correct but he offers no evidence, however, from empirical studies. Software engineers, moreover, work for organisations with substantially different cultures (Wasserman, 1996; Tellioglu and Wagner, 1999). What a software engineer does may depend not only on their knowledge and skills but also on what they are asked to do. The role of the organisation, therefore, also contributes to the success of a project. There have been studies of the important factors, which should be taken into account. As a result of analysing unsuccessful projects, Kavanagh (2000) identified eight important lessons for organisations:

- Analyse and understand full implications of proposed systems
- Specify system taking account of business and user requirements
- Look at scale and complexity to see if project can be broken up into a series of small projects
- Involve senior management
- Provide high quality project management
- Draw up risk management and contingency plans
- Review project
- Plan to train staff

Keil et al. (1998) acknowledge that software projects are notoriously difficult to manage. In a study, experienced software project managers identified what they saw as the main risks to a project. Three panels were set up of practitioners from the USA, Finland and Hong Kong. All three panels identified the same factors even if they attributed varying levels of importance to them. These factors were: lack of top management commitment to the project, failure to gain user commitment, misunderstanding their requirements, lack of adequate user involvement, failure to manage end users expectations, changing scope/objectives, lack of frozen requirements, introduction of new

technology, insufficient/ inappropriate staffing, and conflict between user departments. Overall, the issues for organisations are much the same in both these studies.

**Software Process Models and Associated Notations**

Those who write about the discipline of software engineering describe the lifecycle activities that should be undertaken (Lending and Chervany, 1998; Pfleeger, 1998; Schach, 1999; Sharma and Rei, 2000; Pressman, 2001; Sommerville, 2001). Life cycle activities depend upon the software development process followed such as the waterfall model, prototyping, data-centred or the evolutionary approach. The traditional lifecycle or waterfall model is typically described in a sequential fashion moving from stage to stage involving requirements specification, software design, coding and testing, each requiring a set of deliverables (Sommerville, 2001). Iterations between phases are required, however, to enable revisions to be made whilst an application is under development. The traditional lifecycle is still used in industry particularly for larger systems engineering projects (Sommerville, 2001). In the data centred approach the focus is on the data rather than function and the specific method of entity relationship modelling (Sallis et al., 1995).

Prototyping an application begins with some requirements gathering and moves on to the development of some part of the system. It assumes that the customer can communicate their requirements more effectively once there is something to look at (Yourdon, 1994). The developer may undertake prototyping when there are no clearly stated requirements. It is also appropriate when the developer has little experience of the application domain or of the tools and languages that will be used to develop the system (Yourdon, 1994). The subsequent prototype can be discarded as in throw away prototyping or retained as the basis for further development as in evolutionary or incremental prototyping (Schach, 1999; Pressman, 2001). Problems can arise if the prototype is taken to be a working version of the application.

There are other iterative and evolutionary approaches. Evolutionary in this context means that increasingly complete versions of the software are developed (Pressman, 2001). One such is the spiral model (Boehm, 1988), which uses prototyping to reduce risks as well as incorporating the phases of the classic life cycle (Pressman, 2001). Another evolutionary approach involves component-based development and is centred round the Object-Oriented (O-O) approach. The Unified Process (Jacobson et al., 1999) and Rational Unified Process (Kruchten, 1999) are examples of this.

Currently, there is also a move to using what are termed agile methods (Beck, 1999; Fowler, 2001). Such methodologies "attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff." (Fowler, 2001, p2) These methodologies emphasise the frequent delivery of software, regular (daily) contact with the customer and minimising documentation. Some of the main advantages and disadvantages of this and other software process models (Sallis et al., 1995; Schach, 1999; Fowler, 2001; Pressman, 2001; Sommerville, 2001 ) are shown in Table 1.

Once a life cycle has been selected a software engineer has to make a decision about what notations to use to model and document the system. These make it easier to understand a problem and shape a solution (Robillard et al., 2003). A large number of notations are available including entity relationship, data flow and data structure diagrams (Jordan and Machesky, 1990) and the nine notations specified in the Unified Modelling Language, use case, class, sequence, collaboration, activity, state, component, deployment and package diagrams (Booch et al., 1999). Robillard et al. (2003) observe that gains in uniformity, reliability and productivity are made when notations are consistent across the various phases of the lifecycle. It is important though that staff have tools to support them and are trained in their use.

| Process Model | Characteristics | Advantage | Disadvantage |
|---|---|---|---|
| Iterative waterfall | Iterative | Appropriate when requirements understood | Iterations costly |
| Data centred | Iterative | Development of good data model | Less emphasis on function |
| Prototyping | Evolutionary | Establishes requirements | Danger of prototype becoming final system |
| Component-based | Evolutionary | Software re-use | Comparatively untried |
| Agile methods | Evolutionary | Close involvement with the customer | Not suitable for large projects |

**Table 1: Comparison of process models**

## CASE TOOLS

To assist software engineers to produce high quality software, CASE tools, which automate manual activities, have been developed. CASE tools are available to assist at all stages of the lifecycle and can be used for many tasks (Sommerville, 2001). Pressman (2001) provides a taxonomy of CASE tools by function, identifying 24 different categories (Table 2). They can be as simple as a single tool for supporting particular activities or as complex as a complete environment involving a database, hardware, networks, operating systems, and myriad other components. Those which bring together a suite of tools are referred to as integrated CASE (I-CASE) tools.

| | |
|---|---|
| Business process engineering | Analysis and design |
| Process modelling and management | PRO/SIM |
| Project planning | Interface design and development |
| Risk analysis | Prototyping |
| Project management | Programming |
| Requirements tracing | Web development |
| Metrics and management | Integration and testing |
| Documentation | Static analysis |
| System software | Dynamic analysis |
| Quality assurance | Test management |
| Database management | Client/server testing |
| Software configuration management | Re-engineering |

**Table 2: Classification of CASE tools (based on Pressman, 2001)**

The proposed benefits of I-CASE tools include the following (Pressman, 2001):

- Smooth transfer of information (models, programs, documents) from one tool to another and from one software engineering phase to another.
- Reduction of effort required to perform umbrella activities e.g. configuration, quality assurance and documents produced.
- An increase in project control achieved through better planning, monitoring and communication.
- Improved co-ordination among staff members.

The use of I-CASE tools also creates important challenges for tool developers. Integration demands that relevant information be represented consistently, that standardized interfaces are available on tools, that a homogeneous way of communicating between tools and developers is available and that it is easy to move among versions of operating systems and hardware platforms (Pressman, 2001). I-CASE users often experience a productivity decrease for the first 6 months, and it may take 12 to

18 months before productivity gains are visible. Introduction to any new technology could contribute to this productivity decrease.

Ideally, automated tools should support both lifecycle and project management activities. Sharma and Rei (2000) developed a framework to assess CASE tool usage, which essentially integrates both types of activities. The framework has three processes: production, coordination and organisation. The production process is decomposed into subprocesses related to lifecycle activities such as the representation of objects, relationships and processes, and testing/validation tasks such as test data generation and automatic restructuring of program code. The coordination process comprises the control and cooperation subprocesses. Control is associated with tasks such as the enforcement of policies, resource management and auditing whilst the cooperation subprocess is concerned with enabling people to exchange information (either electronically or face to face). Finally, the organisation process is split into support tasks to assist users to understand and use CASE technology effectively and infrastructure tasks (such as the development of a repository) to enable sharing of knowledge about a project.

Sharma and Rei surveyed over a thousand organisations about their usage of CASE tools. Only a seventh of those who responded had adopted CASE tools. The percentage of those adopters who actually used CASE tools for the purposes of Production, Coordination and Organisation are 68%, 45% and 66% respectively. Overall 60% of tasks were supported to some degree by CASE. The figure for Coordination is surprisingly low. There were also some discrepancies within the categories. There was a very high CASE tool adoption rate for production activities such as the representation of objects, relationships and processes (95% of the respondents) with a much smaller proportion of adopters using CASE for testing and validation purposes (39%). Similarly, within the "Organisation" category, 86% used CASE for infrastructure activities but only 46% for support activities.

Overall, Gray et al. (2000) concluded that it was hard to provide evidence of productivity gains or increases in the quality of the product from the use of CASE tools. Further research into the practices of software engineers was required in order to build usable tools.

**Reasons Behind Non-Use of CASE and I-CASE Tools**

Whilst there are a large number of tools on the market they are not necessarily widely used. There is an extensive literature on the adoption of CASE tools which focuses on the factors that play a part in whether such tools are accepted or not. These can be categorised according to organisational, interface, people and tool related issues. Organisational issues are concerned with cost (Jarzabeck and Huang, 1998; Finnegan et al., 2000), vendor support (McChesney and Glass, 1993; Finnegan et al., 2000) training in the use and benefits of such tools (McChesney and Glass, 1993; Iivari, 1996; Sorensen, 1993), use of appropriate life cycle methodology (Holt, 1997), introducing CASE into the workplace (Misra, 1990; Orlikowski, 1993), compatibility with other tools (Misra, 1990) and imposition of standards, for example making tool use compulsory ((Iivari, 1996). CASE tool functionality has many dimensions. At a high level the issues to be considered include the complexity of the features provided as well as whether the tool supports multiple users and facilitates reverse engineering (Lending and Chervany, 1998; Gray et al., 2000). The smooth transition between the analysis, design and implementation components is another important consideration (Lending and Chervany, 1998). Some more detailed areas relate to the level of customisation available, the speed of access to the data dictionary and the quality of reports (Misra, 1990).

Closely related to the functionality offered by a tool is its usability, that is its ease of use and ease of learning (Lending and Chervany, 1998; Gray et al., 2000). Ease of navigation as well as assistance with layout are concerns that have been mentioned in this regard (Misra, 1990). The provision of adequate graphics primitives, suitable error messages and on-line help have also been discussed. Interestingly, Finnegan et al. (2000), found that people did not find CASE tools easy to use even though they were perfectly happy with the provision of help, ease of navigation etc. This is in line with several other studies (Gray et al., 2000). Users find the notations and editors inadequate for their purpose, proving to be an obstacle rather than an aid. Sommerville (2001) also notes that CASE tools do not offer a great deal of support for members of a team.

There are also factors associated with the needs of the people who have to use such tools. Jarzabek and Huang (1998) consider that CASE tools do not support problem solving and creativity. They believe that if a given methodology is enforced too strictly, software engineers are likely to spend more time fitting their ideas to the methodology rather than actually solving the software problem at hand. A more 'free-style' approach to software engineering, at the initial stages, at least, is likely to inspire software engineers to be more creative and therefore more successful in their endeavours. There is the requirement to build CASE tools that "bridge the conceptual gap between a computer system and human thinking" (p95). Gray et al. (2000) also believe that CASE tools should complement the users' creative problem solving processes. Another important people factor relates to high expectations of CASE tools by users. Such expectations that I-CASE should work on any situation and with any methodology are a major reason for failure (Aaen, 1994; Jarzabek and Huang, 1998; Chmura and Crocket, 1995). Finally, system developers prefer high autonomy but the I-CASE tool does not allow for this (Lending and Chervany, 1998). Developers do not feel motivated. Lending and Chervany conducted a tool usability measurement and discovered that neither intrinsic motivation (tool is fun to use) nor extrinsic motivation (tool is perceived to be useful) was high. Orlikowski (1993) also pointed out that the attributes of people also played a part in developers reaction to CASE tools. She suggested that developers with considerable investment and experience of traditional systems development practices were more reluctant to use CASE tools than those with less time and experience in systems development.

## RESEARCH METHODOLOGY

The objective of the study was a preliminary investigation of the practices of software engineers within New Zealand, with particular reference to their use of development tools. It was decided to employ a qualitative research approach using structured interviews, rather than a quantitative method such as a questionnaire, because of the exploratory nature of the study and the complex processes involved. In particular, interviewing permits the collection of rich data for analysis as well as providing the flexibility for asking follow up questions (Scott, Clayton and Gibson, 1991). The case study method is suitable where there are a large number of variables (Yin, 1993). A multiple rather than a single case approach was chosen to provide greater diversity.

Once the research objectives were determined, the relevant literature on Software Engineering practice and the use of CASE tools was reviewed. This was used as the basis for developing the interview protocol which was subsequently revised after testing it out in a pilot interview. The focus was on determining the activities of the developer and other members of the team on a current project. The final version of the protocol had three sections: general questions to make the participant feel comfortable; in-depth questions on the project lifecycle, management activities and the tools used on a particular project; and, finally, there were some general questions concerning the use of CASE tools. A copy of the questions was sent in advance to the developers.

Purposive or theoretical sampling was used to select the participants (Patton, 1990). Five people were selected who, it was anticipated, would provide rich data about working as software engineers. All of them were required to be currently developing software and using tools. It was also seen as important to cover different environments rather than interview several developers from an organisation. The people selected were working on quite different kinds of projects and for companies of different types and sizes. Three of the developers were male and the other two female. However, to preserve confidentiality for them and their organisation, all the developers are referred to as "he" in this paper.

All the interviews were recorded and transcribed. The transcript and summary were sent to the interviewees for validation. As a result some small changes were made. The data for each case was analysed by inspection of the interviews with results entered into a table for each developer. In an iterative process these results were scrutinised and revised. Summary tables were then derived to show organisational, project and tool usage information. The results for each case could then be compared and contrasted before similarities and differences with the literature were identified.

Threats to reliability and validity were minimised by careful case selection, using multiple researchers, employing purposive sampling, mechanically recording data, having subjects review the interview summaries, scrutinising in detail the results of the analysis and using constructs defined in

the literature (Patton, 1990; Yin 1993). Whilst there were only five cases, as Perry and Staudenmayer (1994, p37) comment, "data on real users, even if the sample is small, is revealing." Fred Brooks (1988), too, notes the importance of obtaining real data on user behaviour for progress in software development to take place.

## RESULTS

Table 3 shows the overall information about the developers and the companies for which they worked. The companies in the study varied widely in size. A company size of 1 to 50 members is regarded as small, 50 to 150 as medium and 150 plus as large, in a New Zealand context.
Four of the developers had extensive experience in the computing industry. Developer A on the other hand had spent only eighteen months working in software development (following on from 5 years university study in IT). He was employed by a commercial software development firm (Organisation A), with a little over twenty staff, which handled new technologies with an emphasis on real time and embedded systems software. The owner of this company suggested Developer A as a candidate for interviewing since, despite his lack of experience, he had a good understanding of the processes and procedures followed by the organisation. This proposal was accepted since it provided the opportunity to obtain data about software engineering practice in a small firm.

| Developer | Employee Title | Employee time in software industry | Size of Company | Generic Area |
|---|---|---|---|---|
| A | Systems Analyst | Limited | Small | Real time and embedded systems |
| B | Analyst/ Programmer | Extensive | Large | Administration |
| C | Development Manager | Extensive | Medium | Business applications |
| D | Senior Software Engineer | Extensive | Large | Administration |
| E | Software Development Manager | Extensive | Medium | Real time and embedded systems |

**Table 3: Developers and their organisations**

The other developers all worked for larger companies. Developers B and D were employed in the IT department of very large companies in New Zealand terms (Organisations B and D respectively) with the responsibility for building administrative systems. Developer C worked on a variety of projects, dictated by the contracts the company obtained for software development. Finally, Developer E was employed in an organisation which delivered real time and embedded systems software. These four people were all in charge of the project they described. Whilst they had to work within the framework of company policies they had some ability to influence decisions.
In this study, the project undertaken by Organisation A was a concept development project where new technology was being applied (Table 4). The applications in Organisations B, and D would be classified in the Pressman typology as new developments, responding to customer requests, that in Organisation C a re-engineering project and in Organisation E an application enhancement. There was also an element of concept development in the project undertaken by Organisation E. The kinds of product development undertaken were one-off contracts (Organisations A and C), in house development (Organisations B and D) and mass production in Organisation E.

| Project Developer | Project size | Project Type | Kind of Development |
|---|---|---|---|
| Developer A | Medium | Concept development | One off contract |
| Developer B | Medium | New development | In house |
| Developer C | Large | Application maintenance | One off contract |
| Developer D | Medium | New development | In house |
| Developer E | Medium | Application enhancement | Mass production |

**Table 4: Projects in organisations**

The size of a project can be categorised, in a New Zealand context, according to the number of people involved with the development process where less than three is small, from three to nine medium, and greater than ten is large (Groves et al., 2000). Given this classification scheme, the only large project was undertaken by Organisation C with all the others medium in size.
The project management and lifecycle activities that were engaged in, together with tools used to support them are described below. This is followed by an extensive discussion of CASE tool issues concerning adoption, selection and integration.

**Project Management Activities and Tool Support**

Project planning and scheduling was undertaken for all five projects. Microsoft Project was used for these purposes in Organisations A, B and D. Gantt charts were seen as particularly useful by Developer D although he also mentioned that the resource allocations could become corrupted.
The other management activities were more extensive in the case of Organisations A, C and E. Company A held the project plan, risk assessment, minutes of meetings, tasks allocated, hours per staff member etc in the "projects database" in Lotus notes. Project reports were produced from this data so that the project coordinator could review progress. NIKU was used in Organisation C to manage the set up of the contract, the conditions, resources, time recording, invoicing etc. Microsoft Project was available in the company for those who preferred to use it for planning. Finally, an in-house project management framework, which provides templates, checklists, role descriptions, good practice guidelines and milestones, was used to direct such activities in Organisation E with relevant information held in Word. Version control software was used on all the projects except that managed by Developer D. Whilst version control software was available within the company, its use was not seen as essential on this medium sized project.
Project management procedures were very strictly enforced in Organisation A. All work tasks were performed within a group environment and the methodology, tools, outputs and related activities were prescribed. These standards were formally monitored and audited on a regular basis. In contrast, Developer B used less formal procedures for developing systems. There were some means of enforcing consistency through the standards agreed by all the team members for project documentation such as data models. Progress was monitored, too, on a regular basis. Much though was left to the individual's preferences with regards to team organisation to deliver systems on time. Procedures were clearly laid out on the project managed by Developer C. These were agreed in advance with the client. It was vital in this case to have clearly defined procedures as several people were working on the project including many from outside the company. There was some latitude, however, with regard to the choice of tools. Where the number of developers on the project were small as in the case of Organisation D there was not quite the same requirements for formal standards to be imposed. Finally, Organisation E had clearly specified procedures although the tools on a project could be changed, within limits. To classify these projects according to the degree of rigour applied to the selection of the task sets, Organisations A, C and E are at the strict end of the spectrum with Organisations B and D towards the structured/casual end.
Various tools were used for project documentation in these organisations including Word, Powerpoint, Excel, and Lotus Notes. The documentation produced was usually extensive except in the case of Developer B who stated that "*Documentation is minimal. The documentation that is most useful probably gets written in the comments of the code*."

**Software Process Models**

Four of the developers (A, C, D and E) followed what they described as an iterative waterfall approach. As one of the developers noted, the same stages (analysis, design etc) apply whether an organisation adopts the waterfall or other approach. In only one case, Organisation D, was this decision made at the discretion of the developer otherwise it was tailored to the needs of the organisation. Developer B used a data-centred methodology in association with the prototyping of customer requirements. Mockups of the system were built and reviewed with customers. Each mockup involved all the phases in the lifecycle. The prototype could be considerably revised once agreement on requirements had been reached. Nearing the completion deadline, additional staff members could be added to work on the coding and delivery phases.

**Lifecycle Activities and Tool Support**

With regard to lifecycle activities undertaken, in all five cases the relevant requirements were obtained and appropriate models developed. A variety of tools were employed to construct the models. Developer A used Visio for drawing Data Flow Diagrams (DFDs) and Entity Relationship Diagrams (ERDs). The creation of ERDS was seen as a central activity by Developer B who used Power Designer for this purpose. Developer B also drew flow charts in Word to assist with customer communication. DFDs and structure charts were sketched by hand if seen as necessary to clarify complex issues. Class, collaboration, component and use cases were developed by Organisation C using Rational Rose. State charts were drawn in Visio as this was the preferred option of staff on the project. Developer D used PowerPoint as well as Visio for drawing relevant diagrams – workflow and interaction diagrams. Early prototyping of the interface was carried out in Frontpage. Finally, class diagrams and state charts were produced for Organisation E initially in Smart Draw but Rational Rose was later used for this purpose.

In Organisation A, programmers developed the code based on a design specification (text and models) stored in Lotus Notes. Code was written in Visual Basic and C++. Version control was provided by CVS. Once ready for alpha testing the code was stored in the "Software release" database in Lotus Notes.

Developer B based his code on his understanding of the system as defined in the ERD models previously drawn. He developed the system in SQL using Ingres Application by Form. Unix tools were available for searching, editing and limited version control. Data was loaded into the database via a spreadsheet. Other team members assisted with coding. They attended project meetings and were aware of the documentation and models developed. A set of specifications in Word incorporating all relevant diagrams was the basis of the coding for Organisation C since Developer C believed that the output from Rational Rose was insufficient to act as a design specification. JBuilder was the environment used for generating Java code with WinCVS for version control. Developer D also ensured that programmers were provided with a specification in Word which included all the important diagrams. VB6 was the language used for coding. In Organisation E, the design documentation had to be completed before release for coding which was carried out using Visual C++/Studio. ClearCase was replaced with CVS for version control purposes by Developer E as it was more flexible and easier to use.

Testing was carried out on all projects. In Organisation A, the test cases were stored in the project database. Developer B set up a user training database for testing. Customers were able to enter their own data in order to check out the system. The whole testing process was very complex in the case of Organisation C. On the project in question, regression testing was carried out manually because the customer had no methodology for this. Rational Robot was available for this, however, within the organisation. Functional testing was also largely manual. If there were no problems, the software was checked by a simple test script. It was then released for testing by an independent team. Organisation C also used software developed in house for performance testing and simulating the loading environment as well as Clear Quest for defect management. The testing process was simpler in the case of Organisation D where test plans were stored in Word/Excel and a defect tracking tool used internally to capture and assign defects. Finally, automated tools were used to a limited extent by Developer E. BoundsChecker was used to go through C++ code to look for

memory leaks whilst Smartbits set up traffic patterns on a network. On the whole though, a simple terminal emulator sufficed for testing purposes.
For a summary of the tools used on these five projects for management and lifecycle activities see Table 5.

**Learning to Use Tools**

Developer A thought that better use could be made of the tools provided in the company and that "everything is a learning curve." Developer B made it clear that there was a long familiarization process associated with PowerDesigner and that staff did not always get the time they needed to come up to speed with it. Completing the project on time had the highest priority. There was not always time for training new staff who joined the project in Organisation C. Instead, they were expected to learn on the job, with guidance provided by experienced team leaders. In Organisation D, the staff were supposed to teach themselves VB6 by reading a specified text. No training for tool use was provided in Organisation E where staff were expected to learn by using tools and accessing information provided by the tool or from the web.

| | **Tools used (Developer A)** | **Tools Used (Developer B)** | **Tools Used (Developer C)** | **Tools Used (Developer D)** | **Tools Used (Developer E)** |
|---|---|---|---|---|---|
| ***Project Management*** | Lotus Notes | | NIKU | | Word (in amework) |
| | Microsoft Project | Microsoft Project | | Microsoft Project | |
| | CVS | Unix tools | Win CVS | | CVS ClearCase |
| **DOCUMENTATION** | Microsoft Word Lotus Notes | Microsoft Word | Microsoft Word | Microsoft Word Powerpoint Excel | Microsoft Excel |
| **ANALYSIS AND DESIGN** | Visio | Power Designer | Rational Rose Visio | Powerpoint Front Page Visio | Rational Rose Smart Draw |
| **PROGRAMMING** | Visual Basic and C environments | Ingres ABF | Jbuilder | VB6 SQL | VisualC++/ |
| **TESTING** | Lotus Notes | Ingres | In house tools Clear Quest | Word Excel In house tool | Bounds Smartbits |

**Table 5: Tools by activities**

**Tool Adoption, Selection and Integration**

Organisation A enforced the usage of CASE tools from the beginning to the end of a project. Developer A believed that their use helped the staff to meet deadlines. The current tools, however, did not support the transition from phase to phase of the lifecycle. Developer A also recognised the complexity of CASE tools and the problem of ensuring that their power could be used effectively. He suggested that the reasons why I-CASE tools might not be used in the organisation included cost, training, major changes to the existing infrastructure, failure to appreciate the benefits of CASE and the need for a suite of products.
The use of CASE tools was not always enforced for developers in Organisation B. A tool might not be used if it was hindering application development. There was a long learning curve, for example, associated with Power Designer. Transition between stages of a project took place manually.

Problems arose because there was no comprehensive data dictionary. Moreover, Developer B commented, without an I-CASE tool repository, an organisation must rely on institutional memory. This made it difficult for anyone to understand the whole system. The developer believed that the lack of I-CASE tools in the organisation could be explained by their cost and the time required to learn them.

Developer C set up an environment for the project where most of the tools were specified but there was some small element of choice eg. staff could use Visio instead of Rational Rose for drawing state charts. Organisation C was one of the two companies in this study to be using I-CASE tools such as Rational Rose. This had been selected in preference to TogetherJ because of its larger company base in New Zealand. Whilst Rational Rose fitted in at a high level with the process they followed during the project, it was used in an agile way and not to integrate information from each phase. There were issues, also, to do with complexity, cost, supposed benefits, and practicability. The large number of features provided made such products in the opinion of Developer C bloated and difficult to use. This then generated a requirement for support. He also wondered whether the usage of Rational Rose warranted the cost since it was being used primarily as a repository for models. Only a small number of its features were being used and it was thought that the same benefits may be obtained by using Visio. The cost of licenses was also an issue. Since the organisation had only been able to afford a limited number of licenses a developer sometimes had to wait to gain access to the software. Developer C saw the benefits of I-CASE with respect to quality and productivity as overblown. He thought it might be possible to set up a good process with simple tools that could provide similar quality and productivity. Even if round trip engineering (involving generation of code from models, reverse engineering etc) was undertaken, Developer C wondered whether it was practical or possible to use an I-CASE tool for this purpose.

In Organisation D, the developer was able to choose (within budgetary constraints) all the tools for use on the project with the exception of VB6 which was the standard IDE in the organisation. I-CASE tools were not being currently used in the organisation for reasons of cost and size of development team. Given, the small number of developers on a project, Developer D maintained that there was only a limited requirement to communicate in an integrated fashion. If a larger project demanded this facility though then I-CASE would be considered with the costs and benefits evaluated.

Finally, in Organisation E there was some scope to change tools, as occurred with the move from ClearCase to CVS. Organisation E was the only other company to use I-CASE. Rational Rose was again the company choice and Developer E was considering whether to purchase a version that supported the development of real time systems. Rational Rose was not used for code generation although it was used for reverse engineering purposes. Unfortunately, large, complex and cluttered class diagrams resulted from this process. There were also issues concerning the flexibility/complexity, cost, and appropriateness of I-CASE. Developer E thought that the preference currently was for flexible tools which could be plugged together as needed by a project, e.g. program editor and compiler, rather than having a single large integrated environment, which would be complex, expensive and resource consuming. He believed that cost was a significant consideration in moving to a new tool since single simple tools tend to be relatively cheap to obtain (US$200 or less) and it was possible to purchase several of them for the cost of a complex integrated tool. Finally, Developer E also stated that, to be successful, a tool must help without getting in the way – hence the shift to CVS, and the caution in moving to an integrated CASE tool. Most software engineers want to be producing code not drawing diagrams.

## DISCUSSION

The results described above will be discussed in the light of the literature on this topic. Important issues relate to the choice of project management activities, the selection of lifecycle and notations, and the use of tools.

**Project Management**

All the projects involved some project management activities (Pressman, 2001; Sommerville, 2001) but the balance between the activities differed significantly between the companies. It is worth considering what factors contributed to the rigorous processes followed in Organisations A, C and E. The project type partly explains this (Pressman, 2001). In the case of Organisations A and E the project involved concept development whilst the project in Organisations C was a demanding re-engineering application.

There were many other reasons though for the strict procedures followed. Two of these organisations were producing software for customers on a one-off basis whilst the third was involved in mass production. Their financial viability depended upon their success in meeting the requirements of customers. Not surprisingly they had extremely centralised procedures for managing a project and enforcing standards. All these companies had a major requirement to track their progress to date and their expenses. Organisations A and E also were set up to deal with the situation where staff worked on multiple projects and the costs had to be assigned to the appropriate account.

Other factors that are said to play a part when deciding upon the set of project management activities include maturity of the applicable technology, performance constraints, embedded versus non-embedded, size of the project, customer/developer communication and staff expertise (Pressman, 2001). Both Organisations A and E were faced with similar problems to do with the use of emerging technologies and the development of embedded systems with performance constraints. They had developed procedures to ensure that projects of this type were properly delivered. Developing an application in JDEE in the case of Organisation C was a difficult undertaking, given the lack of maturity of the technology. The project managed by Developer C was also a large one involving staff from the company, the customer and an independent testing team. This, too, required that rigorous procedures be employed.

Customer/developer communication was also a critical issue when determining the set of project management tasks. Organisations A and C who were both involved in one off contracts made great efforts to ensure that they always had the relevant information about the project available for the customer.

The expertise of staff did not, however, play a part in the selection of project management procedures in Organisations A, C and E. In Organisation A this issue was irrelevant given the highly centralised procedures imposed. The tasks to be carried out were also clearly laid out in Organisations C and E.

As was noted earlier, the degree of rigour with which project management activities were applied by Developers B and D was rated as structured/casual. Even though the standards for documentation were determined by the team in Organisation B, these were still followed (Rada and Craparo, 2000). It is interesting to consider the reasons, though, why the process framework could not be classified in these organisations as completely structured given the importance of the applications. Firstly, the financial pressures were not so great for Developers B and D working in IT departments of large companies to produce in-house software. Whilst in Organisation B there were procedures in place for scheduling activities and reviewing, the documentation produced was limited. Planning could be quite fluid. If a project was falling behind schedule then additional staff members could be added to the programming team. These programmers attended project meetings and were familiar with the documentation. This strategy was possible because staff were available and no extra costs had to be passed on to a client. Secondly, for Developers B and D, since the customer was on site, it was easy to involve representative users in prototyping and testing activities. Thirdly, Developers B and D were both experienced team leaders who were given considerable autonomy. Developer D could also rely on the competence and knowledge of staff who had experience with a variety of languages and tools. Fourthly, Developer D only had small numbers of people to work with at each phase of the project. Microsoft Project sufficed in these circumstances. Version control software was available within the company but its use on this project was not seen as necessary.

Commercial pressures, however, do not wholly explain the difference between the highly structured way in which, for example, Organisation A worked when compared with the more informal work standards of Organisation B. Nor does the nature of the applications (concept development versus

customer request) since the customer requests were of a fairly urgent nature and impacted on the business of the organisation. There were many reasons why a more thorough approach to project management would have been beneficial for Organisation B. It had a large number of staff dedicated to software development (65 as opposed to less than 20 developers in Organisation A) with teams to deliver applications. In addition there were considerable pressures to deliver quality systems in a timely fashion. The reason why it had chosen to implement less strict project management procedures relates partly to a difference in philosophy. Management in Organisation B decided to give autonomy to experienced staff who were expected to bring the project in on time, providing them with additional resources for programming if required. This is essentially a cultural difference between Organisations B and A in particular (Wasserman, 1996; Telleglu and Wagner, 1999).

Advice is given in the literature to assist organisations cope with the difficult task of project management (Keil et al., 1998; Kavanagh, 2000.) Many of these important principles were followed by these organisations. All of them, for example, emphasised the importance of the customer. Developer A worked closely with the customer for instance and Developer C's team worked on the customer site. The main weakness would be with regard to training staff for a particular project (Kavanagh, 2000; Robillard et al., 2003). Developer A thought that better use could be made of tools, implying that training was inadequate. Developer B made it clear that there were time constraints on training. It was only possible for new staff to be given on the job training for the project managed by Developer C, with time again a major limitation. In Organisations D and E staff were expected to upskill themselves. This was seen as possible because the staff employed were believed to be sufficiently well-educated to learn from texts, on-line tutorials etc. Time constraints, therefore, had an impact on the training opportunities offered to staff on a project. It was not always possible to train staff in the use of basic let alone advanced features of tools.

**Software Process Models**

Lifecycle activities are a critical component of the software developers' activities (Pfleeger, 1998; Pressman, 2001; Sommerville 2001). In this study, four of the organisations used an iterative waterfall system tailored essentially to the requirements of the project (D) or the Organisation (A, C and E). There were object-oriented aspects to the projects developed in Organisations C and E (development of use cases, classes, state charts etc) but experience dictated that there was no need to move to a strictly O-O approach given that the lifecycle followed catered to the requirements of the organisation for managing application complexity. Organisation B built administrative systems centred around databases so that a data-centred lifecycle developing ERD diagrams for analysis and design purposes was an appropriate choice in these circumstances (Sallis et al., 1995). To handle the problem of identifying functionality (Sallis et al., 1995), prototyping, usually throwaway, was important for stabilising customer requirements (Schach, 1999). The overall process followed was an iterative one. For none of the projects in these five organisations was an evolutionary process seen as applicable. Perhaps the comparative recency of the Rational Unified approach, for instance, may be an obstacle as there are not enough people with experience using it.

None of the organisations used an Agile methodology (Fowler, 2001) although it was clear that philosophically Organisation B was trying to avoid too much process. Documentation, for instance, was kept to a minimum. The customer was also very much involved in the testing process.

**Lifecycle Activities and Tools Used**

In all the organisations that were studied, well-accepted notations were used to model the system at the analysis and design phases - entity relationship, data flow, data structure, class diagrams, state charts. These were usually drawn using an appropriate package: Power Designer (1 user), Powerpoint (1 user), SmartDraw (1 user) Rational Rose (2 users), and Visio (3 users). The ease of use of Visio for drawing was emphasised by all those who produced analysis and design diagrams (including state charts) with this tool. Developer B's preference for producing some drawings by hand will be discussed in the section on tool adoption. Only Developers C and E who used Rational

Rose for class diagrams on their projects could ensure consistency across the notations used to describe the project as suggested by Robillard et al. (2003).
Developing a complete design specification for the coding phase was an important activity for all but Developer B. Developer E observed that it was company policy that design documents had to be up to date before release for coding and this required effort and discipline when the tools were not integrated. The design specification was also kept for the purposes of customer reference in Organisations A and C who had their clients to satisfy. It has already been noted that, philosophically, Organisation B was committed to less formal procedures than the other companies.
Testing was carried out on the projects described with virtually no use of automated tools. Rational Robot was available in Organisation C but could not be used on the project since at the time, the customer had no methodology for Regression testing. Performance was also checked by software developed in-house in Organisation C. This was useful in both the implementation and maintenance phases of the lifecycle. It was possible to check the application could deal with the likely volume of transactions and ensure that when the system was up and running that the service level agreement was adhered to.

**Comparison of CASE Tool Usage with the Sharma and Rei Framework**

The framework used for CASE tool usage provided by Sharma and Rei (2000) encompasses both life cycle (Production) and project management activities (Coordination, Organisation). All of the organisations used CASE for representation and analysis purposes. CASE tools only played a limited role with regard to testing (performance testing, generation of network traffic) but no organisation used them for automatic generation of tests or for analysing the program structure. With regard to project management, the Coordination activities were supported fully by CASE tools in Organisations A, C and D and to a limited extent in B and D. Finally, all the Organisations used CASE tools for support and infrastructure activities. It can be seen that CASE Tools were not widely used for testing purposes. Interestingly, one of Sharma and Rei's conclusions from their study was that the adoption and infusion levels of CASE was lowest for testing and validation purposes.

**Enforcing Tool Use**

CASE tools were provided for their users by all the companies although only Organisation A made use of them all compulsory (Iivari, 1996). Organisation C in which large applications were usually built on the client's premises set up an environment for each project. In this case, however, staff were at liberty to use the tools they preferred for project planning or drawing state charts. The other organisations on the other hand allowed their very experienced staff more autonomy. It was not mandatory for Developer B or his team to use Power Designer. Developer D was even allowed to choose some of the tools used on a project within budgetary constraints. This was only practicable because of the breadth of experience with tools and environments which existed among the software development group. Developer E was permitted to replace a tool that was seen as inadequate.

**Tool Adoption**

All the developers noted problems for staff in coming to speed with the functionality of the CASE/I-CASE tools. There was a major problem with using Rational Rose for drawing state charts in Organisation C, for example. This is in line with the findings of Lending and Chervany (1998) that such tools can be time-consuming to learn and difficult to use. All the developers had problems moving from one phase of the lifecycle to another, even Developers C and E who were using I-CASE tools. Producing design specifications, therefore, involved integrating diagrams and text either in a Lotus Notes database (A) or in Word (C, D and E).
People factors were also significant. Developer B in order to understand the system and not be constrained by CASE tools chose to draw some models by hand. This is similar to the freestyle approach, advocated for the initial stages of a project by Jarzabek and Huang (1998). Attributes of individuals also played a small role with respect to the use of CASE (Orlikowski, 1993.) Developer

B, the most experienced of those studied and well-versed in traditional methods, showed the least enthusiasm for CASE tools although they were used for some activities. Developer B also enjoyed the high degree of autonomy permitted (Lending and Chervany, 1998).
The CASE tools that were used had to be suitable for their purpose (Lending and Chervany, 1998.) Developer C selected WinCVS as a far more reliable tool than the proposed alternative. Developer E changed to CVS from a much more powerful tool that offered far too many features. If a suitable tool was not available then sometimes one was written in house that was tailored to the company's requirements, such as performance tracking software (Organisation C) and a defect management tool (Organisation D). Finally, the situation also occurred where a tool (Rational Robot) could not be used because it did not fit in with the client's procedures.

**Purchase of I-CASE**

Only Organisations C and E purchased an I-CASE tool. Whilst recognizing these as effective drawing tools, there were some reservations. The cost (Jarzabeck and Huang, 1998; Finnegan et al., 2000) and effectiveness of such tools were questioned. Both developers saw them as overly complex (Lending and Chervany, 1998). Developer C thought that I-CASE had too many features and could be difficult to use. Rational Rose had been used for reverse engineering by Developer E who was not impressed with the overly complex class diagrams produced. In neither of these organisations was the tool used to its full potential (Sorensen, 1993).
Organisational factors played an important role in determining whether or not to purchase I-CASE. Cost (Jarzabeck and Huang 1998) was cited as one of the reasons why I-CASE tools were not purchased. Organisations B and D, however, had they seen a real need for such tools, could easily have afforded them. Other reasons include the time to train staff (McChesney and Glass, 1993) and introducing I-CASE into busy workplaces (Misra, 1990). When several employees have to be brought up to speed with a new technology and possibly a new methodology then problems arise. The infrastructure may be missing and it may be difficult to match preferences for methodologies/languages with suitable tools. There was also a concern expressed by Developer D that the introduction of such a tool would slow down the production of code. The functionality and usability of I-CASE tools were also seen as an obstacle. The lack of flexibility of such tools was seen as a problem by Developer A. There was, as yet, no requirement to support multiple users in Organisation D.

**Claims for I-CASE**

It is interesting to consider Pressman's (2001) claims for I-CASE in the light of the results reported here. Only two organisations had an I-CASE tool and neither used it to integrate models or as a repository. Developer C believed that the system was too large for all its aspects to be defined within a single repository. Developer E thought that a heavy commitment was needed to learn the tool and take advantage of the integrated facilities. On the other hand the project repository of Organisation A in Lotus Notes enabled it to reduce effort to perform umbrella activities such as configuration management, increase its project control and improve co-ordination without I-CASE. The small number of developers on the project headed by Developer D meant that there was not quite the same need to co-ordinate activities. Developer B noted the problem that arose in Organisation B without an I-CASE repository, that is the lack of documentation about what had happened previously.

## FURTHER WORK

The findings reported in this study are the result of only five case studies. Exploratory studies of this type are required, though, so that the software engineering community can start to find out how practitioners actually work. It is essential to find out about real users so that they can be provided with the kinds of tools that they find useful. However, if further progress is to be made towards improved software engineering processes and methods, better computer-aided software engineering (CASE) tool support, and improved training of software engineers, additional research is necessary.

A questionnaire has been piloted and sent out to several hundred software developers in New Zealand.

## CONCLUSION

In this study the practices of five software engineers were analysed with the results compared to the software engineering literature. The projects reported on varied in type, kind of development and, to some extent, size. Project planning and management was undertaken by all the developers studied but the associated task set could be described as rigorous for three companies and structured/casual for the other two. All organisations followed iterative processes, in four cases a waterfall model and in the fifth a prototyping approach. Suitable notations were used but only two organisations benefited from using the same notations for analysis and design. All the companies provided a selection of CASE tools for their staff.

Overall, various choices could be made about project management procedures, life cycle process and tools except in Organisation A where all of these decisions were determined by organisational policies. The fact that Developer A was relatively junior whilst the others managed the projects they described has no impact on the results reported here. Organisation A was highly centralized under its CEO for reasons that have already been outlined. The preferences of the managers had some impact in the other cases. With regard to the software model, Developers B and D were able to choose an approach they thought suited the project. The situation pertaining to tools was more flexible since some degree of choice was permitted to Developers B, C, D and E. Overall, there is an interesting tension in the results reported here between centralised and decentralised processes. On the one hand, one developer was proud of the organisation's centralised project management activities and, on the other hand, there were two developers who enjoyed the autonomy that they were given in developing appropriate processes.

Because of time pressures, training in the use of CASE/I-CASE tools was limited and they were not always used to their full potential. Tools could be rejected if they did not seem to meet the perceived need. In only two of the five organisations was an I-CASE tool used. Cost and other organisational issues such as introducing I-CASE when a team was under pressure to develop systems were seen as obstacles to investment in this area. The preference seemed to be for flexible, light-weight tools.

Despite the various differences reported, all of the participants in this study followed appropriate procedures for developing software. There was no instance where the process was ad hoc. Perhaps this was because no one was involved in small projects. Many factors appeared to play a part in accounting for the differences reported: the project type, the kind of development, the financial situation of the company, the culture of the organisation, the size of the project, the experience of staff, and the developer's preference. Overall, there appears to be a trend to use more highly formalised procedures (Organisations A, C and E) when companies are under considerable commercial pressure and dealing with new technologies. More autonomy (within a defined framework) was granted to Developers B and D who were delivering in house software.

## ACKNOWLEDGEMENT

The authors gratefully acknowledge funding from the New Zealand Foundation for Research, Science and Technology to carry out part of this research.

## REFERENCES

Aaen, I. (1994) Problems in CASE introduction: experiences from user organisations, **Information and Software Technology**, 36, 11, pp 634-654.

Beck, K. (1999) **EXtreme programming eXplained: embrace change**, Reading, MA: Addison-Wesley.

Behforooz, A. & Hudson, F. (1996**) Software Engineering Fundamentals,** New York: Oxford University Press.

Boehm, B (1988) A Spiral Model for Software Development and Enhancement, **Computer**, Vol. 21, 5, pp 61-72.

Booch, G., Rumbaugh, J. & Jacobson, I. (1999**) Unified modeling language user guide,** Reading Mass: Addison-Wesley.

Brooks, F. (1988) Plenary address**, Proc Computer-Human Interface Conference**, New York: ACM Press, pp 1-13.

Chmura, A. & Crocket, D. (1995) What's the proper role for CASE tools? **IEEE Software**, 12 (2), pp 18-19.

Finnigan, D., Kemp, E. & Mehandjiska, D. (2000) Towards an ideal CASE tool, **SMT 2000**, Wollongong, IEEE Computer Society Press, pp 189-197.

Fowler, M. (2001) The new methodology, http://www.martinfowler.com/articles/newMethodology.html, updated Nov 2001.

Gray, J.P., Liu, A. & Scott, L. (2000) Issues in software engineering tool construction**. Information and Software Technology**. 42, pp 73-77.

Groves, L., Nickson, R., Reeve, G., Reeves, S. & Utting, M. (2000) A survey of software requirements specification practices in the New Zealand software industry, Proceedings ASWEC 2000: Australian Software Engineering Conference 2000. (ed.) D. D. Grant. IEEE Computer Society. pp 189-201.

Holt, J. (1997) Current Practice in software engineering: a survey, **Computing and Control Engineering Journal**, 8 (4), pp 167– 172.

Humphreys, W. (1998) **Why Don't They Practice What We Preach?** Carnegie Mellon University: Software Engineering Institute.

IEEE (1993) **IEEE Standards Collection: Software Engineering**, IEEE Standard 610.12 – 1990, IEEE.

Iivari, J. (1996) Why are CASE tools not used? **Communications of the ACM**, 39, 10, pp 94-103.

Jacobson, I., Booch, G. & Rumbaugh, J. (1999) **The Unified Software Development Process**, Reading, Mass : Addison-Wesley.

Jarzabek, S. & Huang, R. (1998) The CASE for user-centered CASE tools, **Communications of the ACM**, 41, 8, pp 93-99.

Jordan, E.W. & Machesky, J.J. (1990) **Systems Development: Requirements, Evaluation, Design, and Implementation**, Boston: PWS-KENT Publishing Company.

Kavanagh, J. (2000) Project Sick-List, **The Computer Bulletin**, March 2000, pp 24 –26.

Keil, M., Cule, P., Lyytinen, K. & Schmidt, R (1998) A framework for identifying software projects, **Communications of the ACM**, Vol. 41, No 11, pp 76-83.

Kruchten, P. (1999) **The Rational Unified Process, An Introduction** (2nd Edition), Reading, Mass: Addison-Wesley.

Lending, D. & Chervany, N.L. (1998) CASE Tools: Understanding the Reasons for Non-Use, **Computer Personnel**, April 1998, pp 13-24.

McChesney, I.R. & Glass, D. (1993) Post-Implementation management of CASE methodology, **European Journal of Information Systems**, 2, 3, pp 201-209.

Misra, S.K. (1990) Analyzing CASE system characteristics: evaluative framework, **Information & Software Technology**, 32, 6, pp 415-422.

Orlikowski, W. J. (1993) CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development, **MISQ**, Vol. 17, No.3, pp 309-340.

Patton, M. Q. (1990) **Qualitative Evaluation Methods**. Beverly Hills, CA: Sage Publications.

Perry, D., Staudenmayer, N. & Votta, L. (1994) People, Organisations and Process Improvement, **IEEE Software**, July, pp 36–45.

Pfleeger, S. L. (1998) **Software Engineering: the production of quality software**, New York: Prentice Hall International.

Pressman, R. S (2001) **Software Engineering: A Practitioner's Approach,** 5th Ed**.,** Singapore: McGraw-Hill.

Rada, R. & Craparo, J. (2000) Standardizing Software Projects, **Communications of the ACM** , Vol. 43, No 12, pp 21-25.

Robillard, P., D'Astous, P., & Kruchten, P. ( 2003) **Software Engineering process with the UPEDU**, Boston: Pearson Education.

Sallis P., Tate, G., & MacDonnell, S. (1995) **Software Engineering Practice, Management, Improvement**, Sydney: Addison-Wesley.

Schach, S. (1999) **Classical and Object-oriented Software Engineering**, New York: McGraw-Hill.

Scott, C.A., Clayton, J.E. and Gibson, E.L. (1991) **A Practical Guide to Knowledge Acquisition**, Reading, Mass: Addison-Wesley.

Sharma, S. & Rei, A. (2000) CASE deployment in IS organisations, **Communications of the ACM**, 43, 1, pp 80-88.

Singer, J., Lethbridge, T., Vinson, N. & Anquetil, N. (1997) An Examination of Software Engineering Work Practices, **Proceedings of CASCON '97, Toronto,** pp 209-223.

Sommerville, I. (2001) **Software Engineering**, Reading, Mass: Pearson.

Sorensen, C. (1993) What influences regular CASE use in Organisations? – An Empirically Based Model, **Scandinavian Journal of Information Systems**, 5, 1, pp 25 –50.

Tellioglu, H. & Wagner, I. (1999) Software Cultures. Exploring Cultural Practices in Managing Heterogeneity within Systems Design, **Communications of the ACM**, Vol. 42, Number 12, Dec. 1999, pp 71-77.

Wasserman, A. I. (1996) Towards a discipline of software engineering, **IEEE**, 13(6), pp 23-31.

Yin, R. K. (1989) **Case study research: Design and methods**, London: Sage Publication.

Yourdon, E (1994): **Object-Orientated Systems Design**, USA: Prentice Hall International Editions.