A Requirements Engineering Method for COTS-Based Systems Development

Cornelius Ncube

Submitted for Examination of Doctor of Philosophy

Centre for Human-Computer Interaction Design
School Of Informatics
City University
London

May 2000

Table Of Contents – Volume 1

Thesis Tables

Thesis Figures

Acknowledgements

Throughout my PhD study, there have been many influences and encouragements, blind alleys, moments of despair and disappointments. During this period of many uncertainties, I have received tremendous support and encouragements from many people. I would like to thank my supervisor, Dr Neil Maiden for his inspiration, dedicated help, encouragement, insight, advice and guidance. Thanks as well to the School of Informatics, City University, for their funding. I also thank various people who provided help and took part in the studies, in particular Suzanne Robertson from Atlantic Systems Guild, Ms Jane Smith from Quintec Associates Ltd, Mr Andrew Moore from Technical Management Consultancy Limited, Daniel Mulhall from Jago Managing Agent and Dr Lamia Jamal-Aldin from Sussex University. Thanks also to my colleagues at the Centre for HCI Design, especially Professor Alistair Sutcliffe, Dr Kulwinder Kaur-Deol, Dr Mark Ennis, Dr Peter Faraday, Dr Helen Sharp, Dr Julia Galliers, Ms Stephanie Wilson, Ms Elizabeth Bromley, Ms Maia Dimitrova, Ms Marina Krumbholz, Ms Raquel Monja, Mr Nedim Dedic, Mr Hu Jiawei. Thanks also to Ms Beverly Copeland, Nigel Mitchem and Pete Boyd and Ms Nonhlanhla Zulu for their various acts of support. Finally, I wish to thank my friends and family for their ongoing support.

Declaration

The author grants the power of discretion to the university library to allow the thesis to be copied in whole or in part without further reference to the author.

Abstract

An increasing number of organisations are procuring off-the-shelf software products from commercial suppliers. However, there has been a lack of methods and software tools for such requirements acquisition, product selection and product procurement. This thesis proposes a new method called PORE (Procurement-Oriented Requirements Engineering) which integrates existing requirements engineering techniques with those from knowledge engineering, feature analysis, multi-criteria decision-making and argumentation approaches to address the lack of guidance for acquiring requirements to enable evaluation and selection of commercial-off-the-shelf (COTS) software. PORE is designed in part from conclusions drawn from real-world case studies of requirements acquisition for complex software product selection. Such studies are reported in this thesis. The PORE method is part goal-driven and part context-driven, in that it exploits models of the candidate COTS software and customer requirements as well as process goals to guide a requirements engineering team. The method's approach and mechanisms is demonstrated using a well-known commercial electronic-mail system. A number of studies are presented to provide validation for the method. These include three studies in three different organisations to select COTS software products and one study of requirements engineering experts to elicit their knowledge. The results from these studies demonstrated that the method is usable and effective. The thesis concludes with a discussion of future work to improve the PORE method and future research directions on requirements engineering for COTS-based systems development.

Chapter 1

Introduction: Requirements Engineering for COTS-Based Development Paradigm

This chapter introduces the research problem and gives the outline of the thesis.

Chapter 1

Overview: COTS-Based Development and Requirements Engineering

1.1 Introduction:

The world of computer systems development is being revolutionised by the use prefabricated, packaged commercial-off-the-shelf (COTS) software products. Despite this interest in COTS software, there is currently no absolutely agreed definition of what constitutes a COTS product. However, Carney & Long (2000) propose a mechanism that characterises COTS software products 'in a reasonably specific manner'. The definition that is used in this thesis is that provided by SEI (1999) which defines a COTS product as software '*that is sold, leased or licensed to the general public; that is available in multiple identical copies and that is used without modification of its internals (i.e. source code); that is supported and evolved by the vendor who returns the intellectual property rights'*. Typical examples of COTS software product under this definition are e-mail packages such as Eudora, Pegasus and Out Look Express; anti-virus systems such as Dr Solomon, F-Prot and Sophos; requirements management tools such as RequisitePro, DOORS and Cradle; enterprise-wide applications such as SAP/R3, BAAN and PeopleSoft.

This use of COTS software products transforms the way organisations develop their IT systems. COTS software products range in size from small stand-alone packages to very large enterprise-wide packages. The COTS product market is growing very fast (Maiden et al. 1999). It is estimated that there are more than 300 COTS vendors competing in the UK market alone. A worldwide market of over $52bn by year 2002 is predicted (Evolving Enterprise, 1998). In 1997 the Fortune Magazine (1997) estimated that 20 000 companies world-wide paid $10bn to COTS vendors and Dataquest (1997) estimated that 41% of these companies have 1-4 COTS packages with 39% of them having more than 5 packages.

Developing systems using COTS products has a number of important benefits to customers (McGrew & Viega, 1999). Some of the proven or predicted benefits are:

- reduction in development time – competing organisations need to keep one step ahead of their competitors, therefore there is a need to produce their computer systems as quickly as possible;
- reduction in lines of code to be written – COTS products provide functionality that developers would otherwise have to write. By employing COTS products, the code that would have been required to implement that functionality is saved;
- reduction in the complexity faced by developers – COTS software products provide abstractions that hide complexities that a developer would otherwise have to tackle.

Not only can the use of COTS software help reduce development time, it can potentially lead to fewer errors in the non-COTS developed portions of the system. As organisations continue to move towards COTS-Based Development (CBD), systems development will become more like traditional manufacturing: developers will code less but design and integrate more (Voas 1998). However, today's computer systems are more complicated than ever and the time pressures to get them done and put into use is greater (McGrew & Viega 1999).

Therefore, in order to reap the benefits of COTS-based development – reduced time-to-market, less coding, more user choice and lower costs, there is a need for the software development industry to rethink their systems development processes and strategies. There is no reason why this industry cannot learn from traditional engineering disciplines like manufacturing, electrical or electronics engineering, where prefabricated components have been used for many years. The use of COTS software products implies the need for new approaches to system development.

However, inspite of the boom in the use of COTS products, most organisations experience major problems in their implementation. Forrester Research (1997) estimated that for every $1 spent on a COTS software package, $9 is spent trying to integrate it, and this integration accounts for 30% of IT development budgets. One reason for these problems is inadequate requirements engineering (acquisition) and COTS product evaluation and selection, although they are rarely recognised or treated as such.

In a COTS-based development process, early evaluation and selection of candidate COTS software products is one of the key aspects of the system development life-cycle. Its success largely depends on the accurate understanding of the capabilities and limitations of the individual candidate products. To achieve this success, the evaluation and selection of the candidate software products must begin at the same time as the acquisition of initial customer requirements. To gain a reasonable level of confidence in the results of the product evaluation and selection process, rigorous methodologies to guide evaluation and selection of products and acquisition of customer requirements are needed. However, there are few methods or tools that guide the requirements acquisition for COTS software evaluation and selection processes. This lack of methods means that there is no systematic process of acquiring customer requirements and expressing them in way that enables effective evaluation and selection of COTS software products. In short, current requirements engineering methods and research do not address COTS-based development issues.

1.2 How current requirements engineering research fails COTS-based systems development

As requirements engineering continues to be an area of growing importance, recent high profile system failures such as the London Ambulance Services in 1992 (Dowell & Finkelstein 1996) have served as examples of system failures due to an inadequate requirements engineering and selection processes. Indeed, Fred Brooks (1987) told us more than a decade ago that '*no other part of the work than requirements engineering, so cripples the results if done wrong*' and this still is and continues to be the case today.

Indeed, it is a widely agreed view in the systems development community that errors generated during the requirements engineering phase are the most expensive to fix (Boehm 1981). Various studies have shown that fixing an error after the system has been delivered costs in orders of magnitude more than fixing it at the requirements engineering stage. Compared to other phases of system development, these symptoms reflect, by a large margin, the lack of adequate approaches to requirements engineering. The implication is that generating error-free requirements has a high cost

leverage and that even small improvements would be worthwhile. This has even more serious consequences in the COTS-based systems development.

This lack of any requirements engineering focus is particularly more surprising given the new opportunities that CBD offers the requirements engineering process. For example, stakeholders or customers often have prior knowledge of candidate products during the requirements acquisition phase, so acquisition can focus on the requirements that can be used to best discriminate between competing COTS products and products can be rejected as relevant new requirements are acquired (Finkelstein et al, 1996). Indeed, the success of any CBD development largely depends on the successful selection of candidate products and the inadequate identification and acquisition of customer requirements can significantly affect the resulting system. In CBD, requirements are the cornerstone for selecting candidate products to be included in the final system. These products are selected according to their degree of compliance to customer requirements. However, most current research largely focuses on integration of selected COTS products (Vigdar et al 1996, Brown et al 1995), architecture and design (Shaw 1996, Garlan et al 1995) and not on how these products are selected in the first place.

There is an implicit assumption that COTS products meet customer's requirements. However the selection of the right product is often a non-trivial task and requires a careful balancing between customer requirements, product functional capabilities and system architecture. These issues are rarely adequately addressed in CBD research. There is little practical process guidance and methods provided to requirements engineers to assist in acquiring requirements for selecting preferred products from the myriad available in the market. Organisations also do not know how to do the selection process. Rather than approaching the selection of COTS products with the attitude 'what COTS products exist in the market and how can we use them' organisations instead, define strict requirements that either exclude the use of COTS or that require large product modifications to satisfy them (Vigder et al. 1996). Defining requirements in too detail establishes an artificially too high baseline on selection and evaluation of products and may lead to unjustified elimination of candidate products that might have provided reasonable subset of the proposed system functionality (Vigder et al. 1996).

One typical example of this problem was a criteria used by a national airline when selecting a voice recognition COTS systems. The criterion was that if a product demonstration by the product supplier went wrong, that product would be automatically rejected and eliminated from the candidate list. One candidate supplier was invited to demonstrate their product to main stakeholder at the airline site. At that time, this particular vendor was the preferred supplier but the demonstration went badly wrong. The airline lost confidence in their product and the product was automatically eliminated without determining whether it met essential customer requirements. The COTS product selection 'often takes place very early in the process where requirements are fuzzy' (Kontio 1996). The problem with COTS software selection is that organisations ignore or don't pay much attention to the importance of customer requirements. This thesis aims to fill in that gap!

1.3 Thesis scope

This thesis addresses requirements engineering for the COTS-Based Development (CBD) paradigm. It focuses on the processes of requirements acquisition and product evaluation/selection. Traditional procurement issues such as invitation to tender and bid assessment are not within the scope of this thesis but issues such as contract production, supplier evaluation/selection and systems procurement management are covered although not in detail. Traditional systems development life-cycle issues such as systems design, development and maintenance are also not explicitly covered. Figure 1.1 shows the focus of the thesis research.

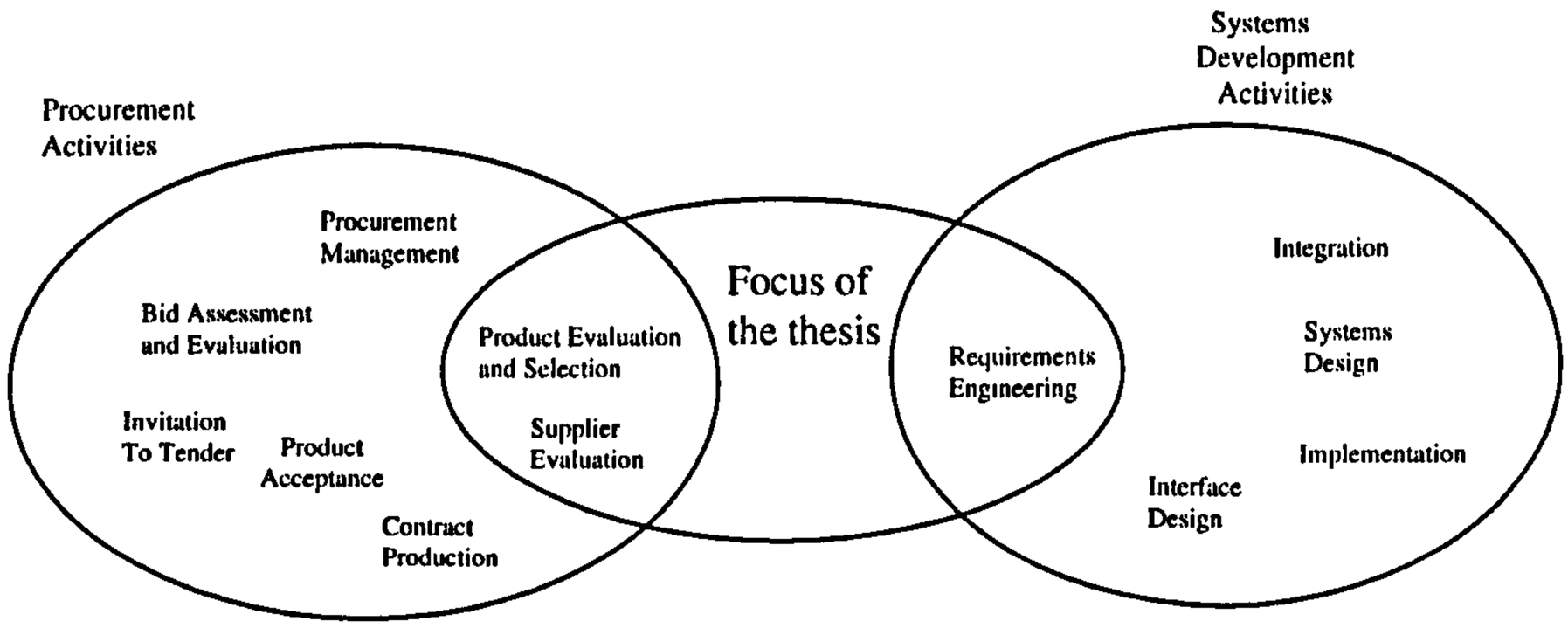


Figure 1.1: Scope of the thesis. Traditional procurement activities such as ITT and bid assessment and systems development activities such as design, implementation and testing are not within the scope of the thesis.

This thesis also:

- Does not address the buy vs. build decisions. Custom building systems allows organisations to ensure that the resulting system exactly meets the customer requirements. On the other hand, building systems using COTS software which is bought off-the-shelf in the market is often cheaper and faster although it may sometimes require that an organisation modify its business process to conform to the application's processes. However, this thesis focuses on buy vs. buy decisions rather than on buy vs. build decisions;
- Does not cover application service providers (ASP). An ASP is a business solution that helps organisations and individuals gain access to software applications via the internet. ASPs aim to meet the needs of business organisations of all sizes that do not have the time, money or resources to purchase, deploy and manage applications. However, lessons learned and techniques developed in this thesis can help organisations and individuals to select a suitable application service provider;
- Does not address software reuse. Software reuse can be characterised as the process of building or updating software systems using existing source code. During software re-use, developers have access to in-house source code with similar functionality within an application domain. In CBD developers do not have access to the software product's source code. The COTS product is used as is and once developers have access or modify the product's internal, its no longer a COTS product and they loose all contractual arrangements;
- Does not handle COTS software product evolution. Product evolution can be characterised as the process of adding new features to successive versions of the product or to legacy software. COTS software products are supported and evolved by the vendor who returns the intellectual property rights. Product evolution is therefore beyond the scope of this thesis.
- Does not address multiple COTS selection. This thesis focuses on the selection of a single COTS software product. The rational for this is that there is a need to first

understand the problems encountered in a single product selection before considering problems of multiple product selection. Also, the problems of multiple COTS selection are too big to be covered as part of a PhD study and are worth another major research undertaking of their own. For these reasons, multiple selection is not within this thesis' scope.

1.4 Thesis objectives and hypotheses

The main objective of this thesis is to develop a requirements engineering method for CBD and the associated processes, techniques, models for guiding the process, and guidelines for requirements engineering and COTS software product selection and to evaluate it in real-world product selection situations. The 6 hypotheses that structure the research are:

H1 New problems arise in the requirements engineering-related phases of CBD that are not addressed in current requirements engineering research.

H2 It is possible to design more effective methods by directly addressing current problems in the requirements engineering phases for COTS-based development;

H3 This method's guidance can be applied in part or in whole to real-world software product selection tasks;

H4 This more effective method can form an essential part of a successful product selection task;

H5 The method's guidance is perceived to be useful and usable by people involved in the product selection task;

H6 The method's advice is at least as good as current expert advice.

1.5 Research contributions

This thesis has implications for requirements engineering for COTS-Based Development process:

- a method, PORE, is proposed to fill the gap in requirements engineering methods and to provide guidance in the use of customer requirements in COTS software evaluation and in helping people choose or select software packages;
- an interleaved requirements acquisition and COTS software product evaluation/selection process model is proposed;
- a process model that identifies key decisions points that should be made in any CBD process is defined;
- five generic processes for achieving each decision point and a sequence for undertaking each process is defined;
- a software product model, a requirement model and compliance model that models compliance relationships between customer requirement and product features are proposed;
- strategies for guiding the CBD process using models, goals and process situations are defined;
- a prototype process advisor tool for guiding a requirements engineering team is developed.

1.6 Thesis outline

The remaining six chapters describe how the aims of the research were met. Chapter 2 reviews related work. It discusses the lack of requirements engineering approaches for CBD and argues for the need of new methods, process models and techniques. Chapter 2 also describes other CBD and requirements engineering research. It concludes with an assertion that there is currently a lack of relevant theories and methods for CBD.

Chapter 3 reports two studies that were carried out to investigate hypothesis H1, which states that 'new requirements engineering problems arise in CBD that are not addressed in current requirements engineering research'. The first was an empirical

study carried out in 3 different organisations. In total, 21 hours of interviews were undertaken. The second was a case study that involved selecting and recommending COTS requirements management system to a customer. The problems identified and lessons learned during these studies served as the conceptual origins of the PORE method that addresses hypothesis H2.

Chapter 4 further investigates hypothesis H2 and describes the need for process advice and guidance in COTS-based development process. It proposes a method that utilises an iterative process of requirements acquisition and product selection and situated process guidance for guiding the requirements engineering team during requirements acquisition and product selection. The chapter concludes by recommending a software tool for handling the large number of process situations that may arise.

Chapter 5 further elaborates hypothesis H2 and outlines the design and implementation of the process advisor prototype to deliver the requirements engineering team with process guidance and advice. The chapter also describes why a tool is needed. The tool acts as a collaborative advisor to the requirements engineers and is designed to integrate with existing requirements management tools.

Chapter 6 tests hypotheses H3 – H6 and describes the PORE method evaluation results. The evaluation is divided into two parts. Part 1 tests hypotheses H3 –H5 and describes case studies in 3 organisations that used the PORE method to procure COTS software products. Part 2 tests hypothesis H6 and describes expert evaluation of PORE's process advice and guidance. Two requirements engineering experts were presented with 7 process 'situations' and asked to describe what they would do and what techniques they would use to solve the situation. The experts' results were compared with predicted advice and guidance provided by the method.

Finally, Chapter 7 summarises the research and contributions described in this thesis and propose future work needed to further refine PORE. This chapter also discusses why multiple COTS software product selection is beyond the scope of the method; why the PORE method should include supplier's view; the buy vs build argument and how lessons and techniques developed in this thesis can be applied in choosing

application service providers. The chapter concludes with a vision for possible requirements engineering for COTS-Based Development future research directions. Figure 1.2 shows the structure of the thesis.

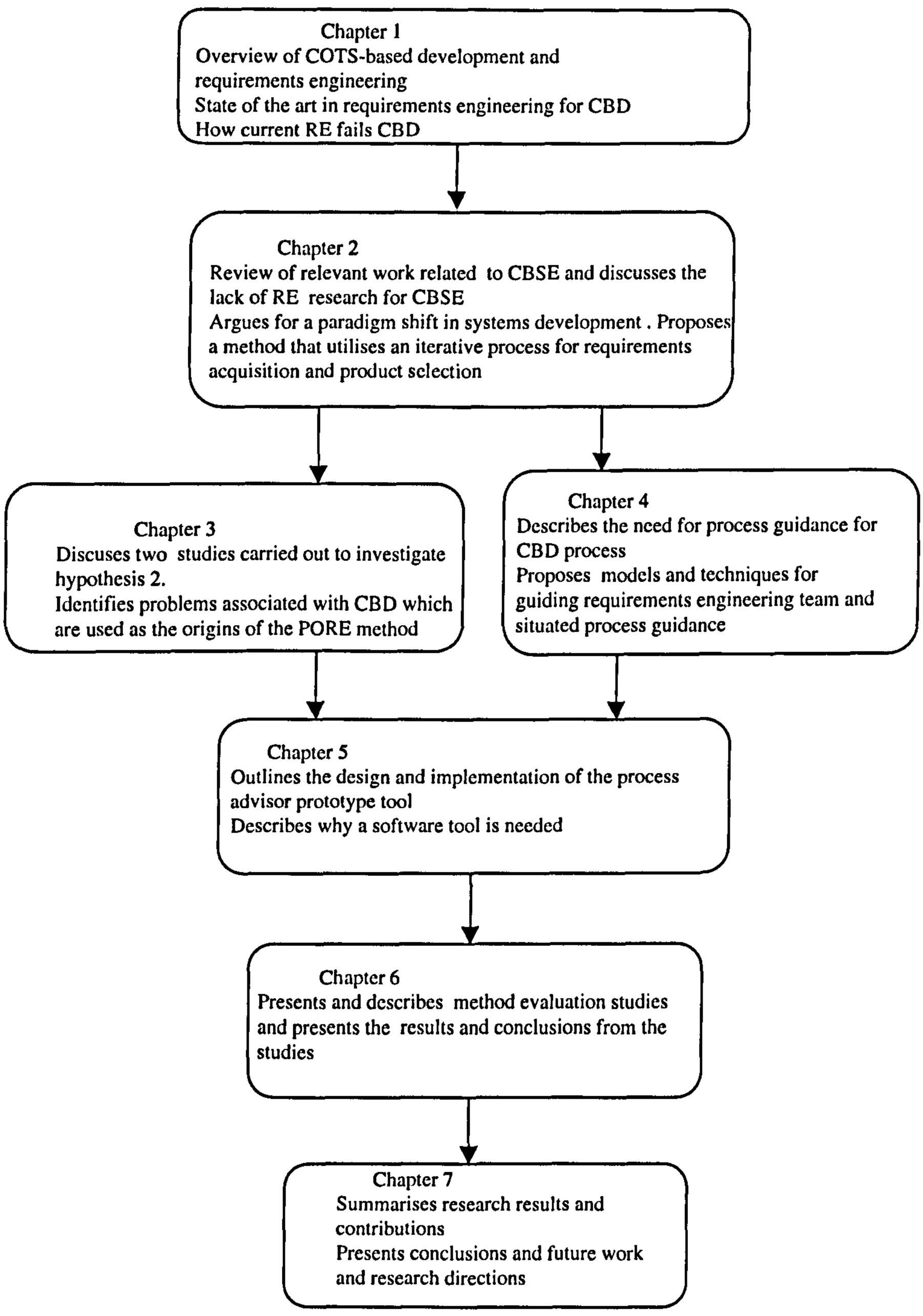


Figure 1.2: The structure and relationships between the thesis chapters

Chapter 2

Current State-of-The-Art and Trends in COTS-Based Systems Development

This chapter provides background knowledge for the thesis and details relevant research.

Chapter 2

Current State-of-The-Art and Trends in COTS-Based Systems Development

2.1 Introduction

This chapter discusses the current state-of-the-art and trends in software packaged-based systems development. Firstly, a brief statement of the problem for adopting the packaged-based development paradigm is made. Relevant literature associated with packaged-based systems development, requirements engineering and decision-making techniques is discussed. This then leads into an identification of current trends in packaged-based systems development research and the lack of recognition of the importance of requirements engineering. The chapter concludes by identifying areas of theoretical and empirical weakness in requirements engineering and decision-making techniques for package-based systems development.

2.2 The problem and its setting

As modern complex software systems become more expensive, organisations are increasingly shifting their system development processes away from bespoke development to package-based systems development. Cheaper packaged software products that can be purchased off-the-shelf and integrated into systems to perform most required functionality are now available in the market. Extensive application packages that satisfy most of the customer's requirements can be now purchased and tailored to meet the customer's needs. Organisations view the use of packaged products as having the potential to reduce the cost and time to develop and deploy software intensive systems. Oberndorf (1999) suggests that organisations that adopt the packaged-based systems development paradigm are '*attracted and motivated by the prospect of adopting best commercial practices, leverage of commercial investments and new technologies*'. However, the success of packaged-based systems development largely depends on the successful selection of software products that meet essential customer requirements. Given the complexities of today's software

intensive systems, the cost and risk of selecting the wrong software product due to inadequate requirements acquisition and product evaluation processes is large. However, despite the recognised importance of requirements engineering to software product evaluation and selection, this importance is not reflected in current research trends for packaged-based development.

2.3 The shift to packaged-based systems development paradigm

Paul's (1994) observation of traditional system development paradigms concluded by stating that these systems development paradigms '*usually lead to systems that are built for one hypothetical point in time and thus confronting users with a paradox of static systems designed to function in a dynamic environment*'. This observation further states that these traditional development approaches are inadequate for the modern complex dynamic systems, for they are '*inherently static in their nature*' and are not adaptive enough for the today's constantly changing environment. Paul further states that today's organisational factors and environments have an '*infinite time horizon*' and change more rapidly than systems can be '*planned, developed and implemented*' using the traditional approaches which are '*finite time horizon driven*' due to their project-based nature. The results of traditional systems development are systems that are forced to adapt to changing circumstances. One of the fundamental problems with traditional development approaches is that they require systems to be built to exact specifications, thereby resulting in systems that are built for '*one hypothetical point in time*', whereas those same systems are required, or expected, to work over some '*time continuum*', i.e. in a continuously changing business environment. In today's dynamic systems environment, this is guaranteed to cause major problems and '*user disappointments*'.

Figure 2.1 below depicts the paradigm shift from traditional development approaches to packaged-based systems development. In the traditional development paradigm, 80% of the system is custom built and only 20% is packaged-developed. In contrast, in the packaged-based development paradigm, 80% of the system is packaged-developed and only 20% is custom built. In the packaged-based development paradigm, the entire systems development process – *requirements acquisition, design, implementation, maintenance* – undergoes radical change (Carney, 1998). In a

traditional development process like the waterfall, all requirements come first. In the packaged-based approach however, requirements are defined iteratively and many requirements only become known after several iterations, as the system is being developed.

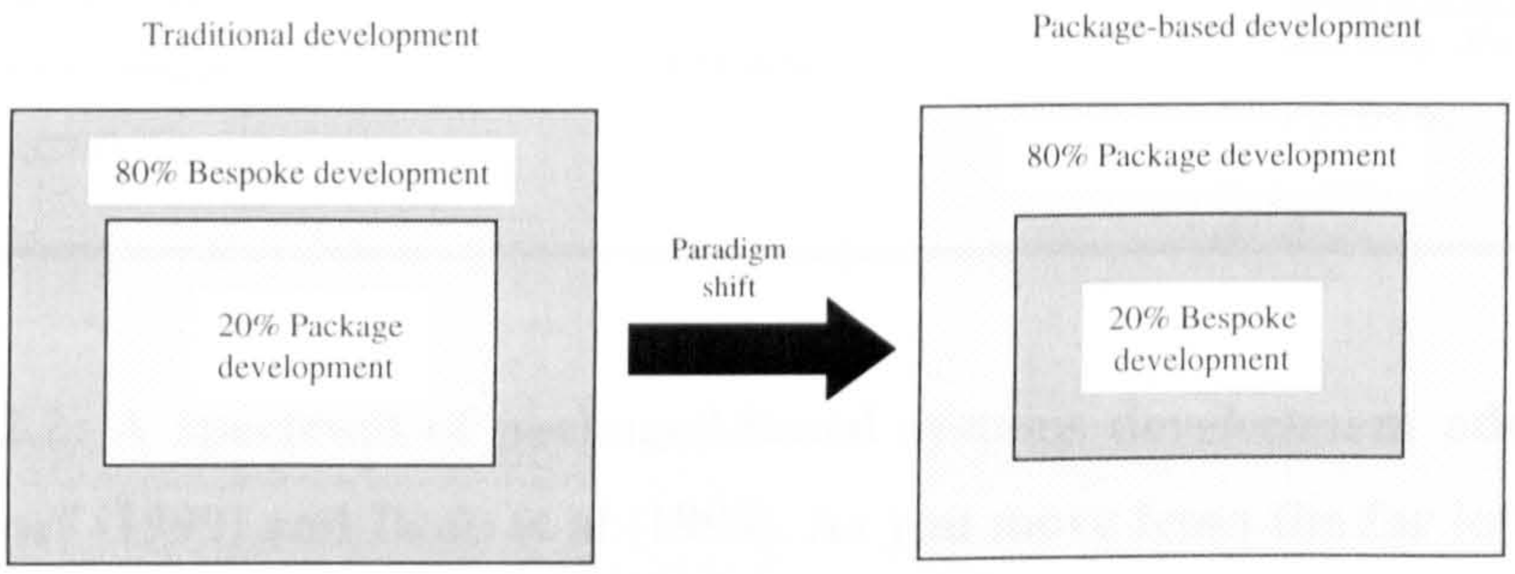


Figure 2.1: Paradigm shift from bespoke custom-built systems to packaged-based development, (Dean et al 1997). In the traditional development most of the system functionality is custom built and only a small part of the system is purchased of the shelf. In the packaged-based development, most of the system functionality is procured off-the-shelf and only the system functionality that is unique to the customer is bespoke built.

2.4 Current trends in packaged-based systems development

Packaged-based development covers COTS-Based Development (CBD), Component-Based Software Engineering (CBSE) or even Enterprise Resource Planning (ERP) package development. However, there are subtle differences between these forms of packaged-based development and their perceived meaning. Currently there are no generally agreed definitions. However, a closer look at the packaged-based development concept, current state-of-the art and research trends reveals that the concepts fall into three types – *Enterprise Resource Planning development, COTS-based development and Component-based development* - as indicated in figure 2.2 below (Oberndorf 1999, Dean et al 1998).

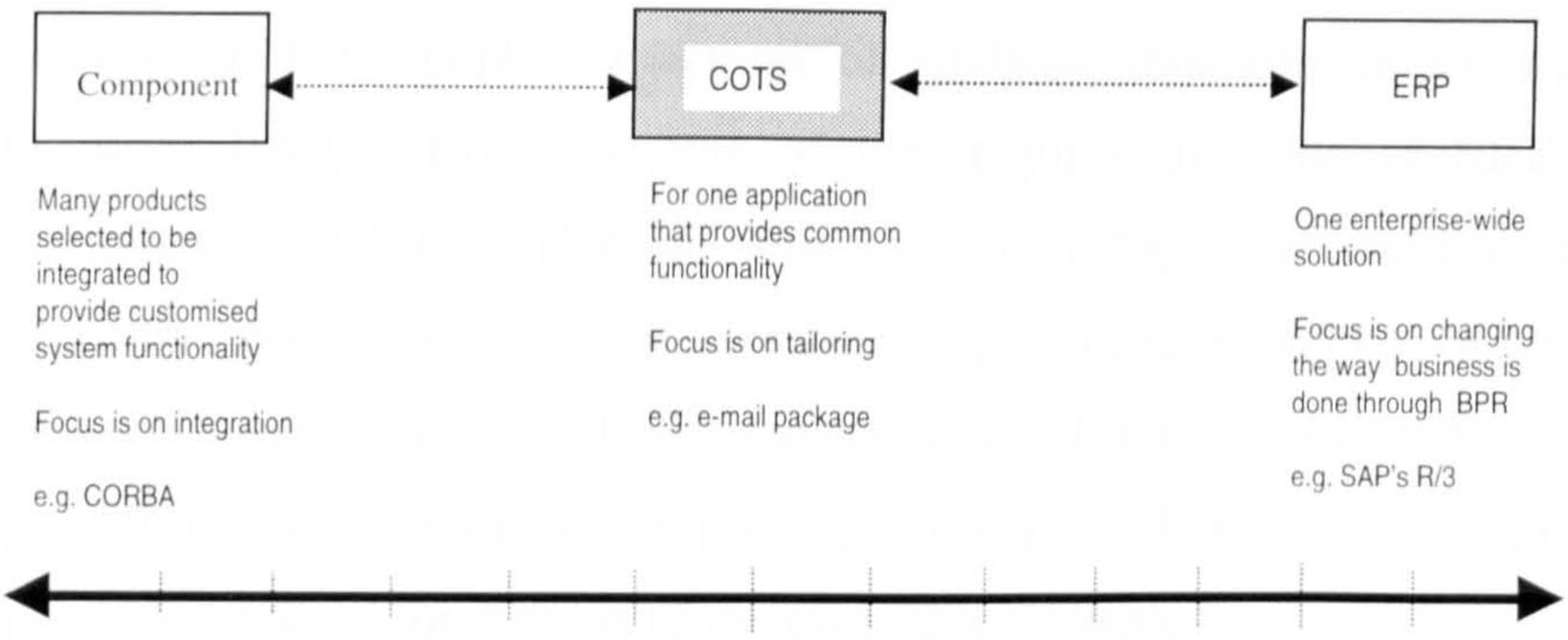


Figure 2.2: A spectrum of packaged-based systems development adopted from Oberndorf (1999) and Dean et al (1998). As you move from the far left hand side of the spectrum to the far right, the size of the packages generally increases and vice versa. However, it is not always ease to explicitly determine the boundaries between the components, COTS and ERP, i.e. when COTS stops and an ERP begins or when a component stops and a COTS start?

The following sections briefly discuss each concept depicted in figure 2.2. Section 2.4.1 discusses the use of ERPs, section 2.4.2 discusses component-based development (CBSE) and section 2.4.3 discusses COTS-based development (CBD).

2.4.1 Enterprise Resource Planning (ERP) systems

The world of business systems development for large organisations is being revolutionised by the use of enterprise-wide, off-the-shelf packaged application software solutions. These very large, integrated packages are designed to transform the way organisations achieve their business objectives through IT. Today, these packages automate business activities for half of the world's top 500 companies and their market has been estimated by Evolving Enterprise (1998) to be well over $10billion and growing. Evolving Enterprise (1998) further estimates that currently, there are more than 300 ERP vendors competing in the UK market alone and predicts a world-wide ERP market of more than $52bn by the year 2002. In 1997 the Fortune Magazine (1997) estimated that 20 000 companies world-wide paid $10bn to ERP vendors, and Dataquest (1997) estimated that 41% of these companies have 1-4 ERP packages with 39% of them having more than 5 packages. ERPs focus on changing

the way business is done through business process re-engineering (BPR) by providing functionality for the complete spectrum of business functions in an organisation (Brinkkemper 1999). However, one of the major drawbacks of ERP's is that organisations that wish to implement them have the prospect of undergoing a major global business process and system re-engineering in order to adhere to their precise processes. Forrester Research (1997) estimates that for every $1 spent on an ERP package, $9 is spent trying to integrate or configure it, and that integration and configuration accounts for 30% of IT development budgets.

Developing systems from ERP packages requires higher levels of organisational change than do other types of package development. Organisations have to change their business processes, organisational structures and business strategies. Davenport (1996) states that '*organisations that do not make the required changes have been faced with the prospect of ending up with a failed implementation after spending large sums of money, time and resources*'. A broad distinguishing feature of ERP packages is that they are comprehensive, highly integrated, complex systems that are very difficult if not impossible to modify in order to support the already existing set of an organisation's business processes without running into major difficulties.

It can be very expensive, time-consuming and impractical to tailor an ERP system. External consultants who tailor ERP systems often have insufficient understanding of the user organisation's business requirements nor the budget or time to understand the organisation's current processes and future requirements. Further more, implementing an ERP system is a very long process and there are few methods or software tools to guide the evaluation or configuration of these packages. The direct consequence of this lack of methods, tools and process guidance is that most ERP implementations disappoint and fail to adequately meet customer requirements when first installed.

2.4.2 Component-Based Software Engineering (CBSE)

Grundy (1999) defines the component-based software engineering (CBSE) development process as '*the process of building applications from discrete, inter-related software components that are often dynamically plugged into running applications and reconfigured by end users or by other components*'. Brown & Short

(1997) define a component as '*an independently deliverable set of reusable services*'. By '*reusable services*' this definition implies that components have capabilities that other components may wish to use. In order for this to happen, a component must have a specification that describes what it does and how it will behave when its services are required by other components. The other important element of this definition is the concept of '*independent delivery*'. Independent delivery refers to the context awareness of a component, i.e. independently deliverable components should typically not be aware of the context in which they are being used. The implication is that components are expected to collaborate with one another to accomplish a solution. From this point of view, Brown & Short's definition is much closer to Grundy's definition of component-based software engineering. However, currently there is no general agreement over what constitutes a component, although different definitions are emerging.

In the summary report of the 1st International Workshop on Component-Based Software Engineering, Brown & Wallanau (1998) put forward three representative definitions that define a component:

- a non-trivial, independent and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture.
- a run-time, dynamically bindable software package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at run-time;
- a unit of composition with contractually specified interfaces and explicit context dependencies only and can be deployed independently and is subject to composition by a third party.

Ning (1999) further defines a component as '*an encapsulated, distributable and executable piece of software that provides and receives services through well-defined interfaces*'. Ning and Brown & Wallnau's definitions seem to agree with that of Grundy and Brown & Short. Although these definitions seem to describe approximately the same concept, the key feature that seems to characterise a

component appears to be the notion of 'component autonomy', i.e. the ability of a component to be deployed or to execute independently.

Current research in component-based software engineering focuses mainly on component infrastructure capabilities and middleware solutions for connecting components in order to provide system functionality and communication among the components (Brown 1998). A number of component infrastructure technologies have been developed and there seem to be three specific infrastructures on which some measure of standardisation is beginning to occur and for which many components, tools and methods are now available:

- the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA);
- Sun's Java Beans and Enterprise Java Beans;
- Microsoft's Common Object Model (COM) and Distributed Common Object Model (DCOM).

Each of these component infrastructure approaches relies on underlying services to provide the communication and co-ordination necessary to construct applications. The component infrastructures act as the 'road map' that allows components to communicate and to share an understanding of how to use the infrastructures. These infrastructures enable components to be easily replaced by other components offering new or enhanced functionality (Schmidt & Assmann, 1998) whenever they become available. Figure 2.3 depicts the impact of components in Component-Based Software Engineering.

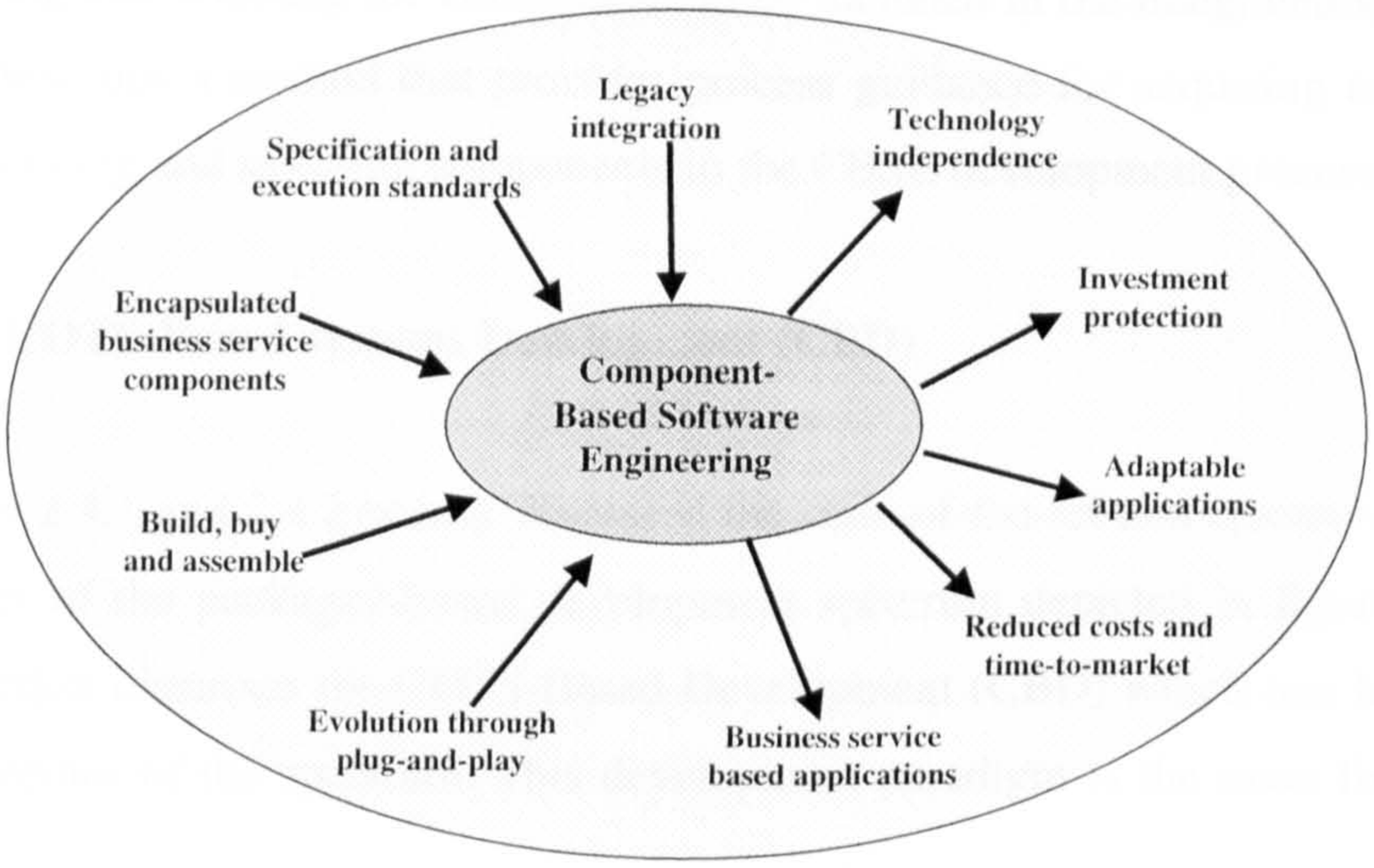


Figure 2.3: Impact of components in the Component-Based Software Engineering development process. Adapted from the Butler Group, September 1998.

Although component-based software engineering is usually associated with specific technologies such as CORBA, ActiveX/COM/DCOM or JavaBeans (Kruchten 1998), CBSE also describes the creation and deployment of software-intensive systems assembled from components as well as the development of such components. CBSE largely involves crafting the right set of primitive components from which to build families of systems. Another current and popular view of CBSE is that components are binary units that are independently produced, acquired and deployed, but that interact to form a functioning system. This assumption that software components are binary units is further enforced by current popular component-based technologies (such as COM, CORBA and JavaBeans) that support encapsulation of so-called binary components (Ning 1999, Wallnau 1998).

A closer inspection of current research reveals that component-based software engineering is more about the development and integration of 'run-time dynamic bindable' binary software components than the integration of medium to large scale packages such ERP systems. The CBSE development process is a developer-centric paradigm and most requirements engineering is mainly focused on requirements for developing the components (e.g. Grundy 1999) and not on requirements for

evaluating and selecting the components to be included in the integrated system. This thesis develops a method that provides process guidance for acquiring requirements for evaluating and selecting components in the CBSE development process.

2.4.3 COTS-Based systems Development (CBD)

Sections 2.4.1 and 2.4.2 briefly discussed the state-of-the-art and research at the two extremes of the packaged-based development spectrum depicted in figure 2.1. This next section discusses the COTS-Based Development (CBD) which lies between the two extremes of the spectrum. This development paradigm is the main focus of this thesis.

As with components, there is no generally agreed definition of what constitutes a COTS product. Various attempts have been made to try to provide a general definition of a COTS product (e.g. Oberndorf 1997, Vigder et al 1997, Kontio 1996, Brown et al 1995). The definition of a COTS product that this thesis adopts is one that is suggested by the Software Engineering Institute (SEI) COTS-Based Initiative which defines a COTS product as:

' a product that is sold, leased, or licensed to the general public; that is offered by a vendor trying to profit from it; that is supported and evolved by the vendor who retains the intellectual property rights; that is available in multiple, identical copies; that is used without modification of its internals' (SEI 1999).

A COTS software product is an application that provides common functionality as opposed to an ERP package, that aims to provide an enterprise wide solution. An e-mail package is a typical example of a COTS product. However, as shown in figure 2.2, at the extremes, it is difficult to distinguish between COTS and ERP packages.

2.5 Current requirements engineering trends

'When the Software Crisis was discovered in the 1960s, considerable effort was directed at finding the causes of the problem' states Dorfman (1999). The investigations that followed the discovery of the 'software crisis' determined that

requirements deficiencies were among the 'most important contributions to the problem' and that requirements 'inadequencies' play a major and expensive role in many project failures (Dorfman 1999). The process of discovering 'requirements' is referred to as requirements engineering. There have been many definitions of requirements engineering that have been proposed. Zave (1994) defines requirements engineering as 'that *branch of systems engineering* that is concerned with the *real-world goals*, *services* provided by, and *constraints* on large and complex software intensive systems. It is also concerned with the relationship of these factors to precise *specifications* of the *system behaviour*, and to their evolution over time across *systems families*'. Costello & Liu (1994) further define requirements engineering as 'the process of *conceptualising*, *specifying* and *validating* the *specification* of the required *behaviour* of the system'. Gause & Weinberg (1989) define requirements engineering as 'the part of system development in which people attempt to discover *what* is desired' while Sommerville (1992) define requirements engineering as 'the process of establishing the *services* the system should provide and the *constraints* under which it must operate'. Another relevant definition is that offered by Davies (1990) who defines requirements engineering as 'the *analysis* and *documentation* of both *user needs* and the *external behaviour* of the system to be built'. Jarke et al. (1993) define requirements engineering as 'embedding systems within their environment rather than on the prescription of the system's functionality and structure'. This definition stresses the distinction between requirements engineering and other phases of systems development.

Different requirements engineering methods have emerged over the years to try and "alleviate" the problems. Dorfman (1999) divides the requirements engineering methods that have emerged roughly into four categories:

- ***process oriented methods*** that take the primary viewpoint of the way the system transforms inputs into outputs with less emphasis on the data itself and control aspects. Examples are Structured Analysis, (SA), structured analysis and design technique (SADT), SSADM, formal methods such as VDM and Z;

- **data oriented methods** that emphasise the system state as a data structure, primary examples are Entity Relationship modelling and JSD;
- **control-oriented methods** that emphasise synchronisation, deadlock, exclusion, concurrence and process activation and deactivation. Primary examples are flowcharting;
- **object-oriented methods** that base requirements analysis on classes of objects of the system and their interactions with each other.

Recent high profile system failures such as the London Ambulance Services in 1992 (Dowell & Finkelstein, 1996) have served as examples of system failures due to an inadequate requirements engineering process. It is a widely agreed view in the computing community that perhaps the greatest outstanding problems in the development of software intensive systems lie in the area of requirements engineering. Indeed, Fred Brooks (1987) told us more than a decade ago that '*no other part of the work than requirements engineering, so cripples the results if done wrong*' and this still is and continues to be the case today. It is a generally agreed view that errors generated during the requirements engineering phase are the most expensive to fix. Indeed various studies have shown that fixing an error after the system has been delivered costs in orders of magnitude more than fixing it at the requirements engineering stage. Compared to other phases of system development, these symptoms reflect, by a large margin, the lack of adequate approaches to requirements engineering. Their implication is that generating error-free requirements has a high cost leverage and that even small improvements would be worthwhile.

Although complex and continually changing requirements is not a unique phenomenon to packaged-based systems development, market volatility and requirements instability have more severe consequences in packaged-based development than in traditional development. Basically there are three types of situations that can cause instability (Watts 1989):

- Unknown requirements in which the users think they know what they want but discover during the initial evaluation of the candidate products that their real needs are not what they had thought;

- Unstable requirements in which while the users may know their general requirements, their specific requirements might remain fluid until much later in the development process;
- Misunderstood requirements in which even though the requirements are known and stable the customers and requirements engineers do not understand them in great detail. The requirements may be misunderstood because of the large size of the system, or because the system is so complex (e.g. a submarine or warship) that it is difficult for the requirements engineers or customers to focus their attention on one aspect of the system at a time, or to visualise the perceived interactions between system functionality.

Another major primary cause of volatility in the CBD is that not only do the user needs change during the time it takes to develop the system, the COTS products themselves may change due to the supplier releasing a new version of the product into the market. Indeed the user's needs may mature because of an increased knowledge brought on by more understanding of the products. Their needs may even shift to a new set of requirements because of unforeseen organisational, environmental or market pressures (e.g. such as new competition or new technology). Another cause of volatility is that requirements are a product of the contributions of many individuals who may have conflicting goals and needs. As this thesis posits, a parallel iterative process of requirements acquisition and product evaluation and selection can address most of the problems of volatility and instability. The notion of a life-cycle with requirements acquisition completed before the design stage can not deal with the twin problem of volatility and instability and therefore not satisfactory for the COTS-based systems development paradigm.

Most requirements in COTS-based development will only become known after initial product evaluation and as the system is being developed or products are integrated. This is especially true when multiple COTS products are used in the development of the system since their interactions will have significant influence on the eventual products to be selected and the overall design of the system (Carney 1998). Also some COTS products impose additional requirements on the system and these 'derived'

requirements are usually unforeseen, although no less important than the original requirements. Therefore, when several COTS products are to be included in a system, it is possible that some requirements or product capabilities can be loosened without substantially affecting the system functionality or its performance (Carney 1998). This means that not only many requirements or required product capabilities can be left undefined, but that they can be left undefined until fairly late in the development process (Finkelstein et al 1997, Vigder 1997) when available product capabilities are known. This can only be achieved by an iterative process of requirements acquisition, product evaluation and selection, and system architecture definition and design, as this thesis suggests.

2.6 Current problems with COTS-based development

Although building systems from COTS products offers organisations the opportunity to reduce the development time and cost of software systems (Oberndorf 1997), there are still many problems that organisations need to overcome. For example, in a COTS-intensive system, many products from different vendors have to be integrated and tailored to provide complete system functionality. In many cases these COTS products will be developed at different times, by different vendors or suppliers with many different styles of use in mind. Organisations will have very limited access to product's internal design and its pre-defined options for customising its behaviour. Customers cannot influence the release cycle of new versions and are left to rely on the long-term viability, integrity and ability of the product's producer (Brown & Wallnau 1998). Also, the life-cycle of the individual products is in the supplier's hands. As COTS products are typical living systems (i.e. they are born, breath and eventual die), their updates, revisions, changes to their internal architecture and the decision to stop supporting them are determined by the product's vendor (Carney 1999) and not by product user. Therefore in assembling these COTS products into an integrated business system, organisations are placed in a situation over which they have no control. However, the impact of some the mentioned problems can be minimised if adequate attention is paid to the process of requirements acquisition and product evaluation and selection.

2.6.1 Why COTS software selection is problematic

Successful selection and effective integration of COTS products that meet customer requirements is problematic for a number of reasons:

- lack of defined process. Most organisations are under pressure to develop systems faster and cheaper, and do not use well defined repeatable processes (Kontio 1996);
- previous lessons are not learned. The lack of well defined processes makes planning and the use of appropriate evaluation methods, techniques and tools difficult. As a result, lessons from previous experiences are not learnt (Kontio 1996);
- evaluation criteria are sometimes vague and open to different evaluators' interpretations (Kontio 1996);
- evaluators tend to focus on the product's technical capabilities at the expense of the non-technical or soft factors such as business issues, supplier issues and contractual issues (Powell et al 1997);
- lack of access to COTS product's internals due to their black-box nature makes it difficult to understand them and makes evaluation hard (Dean 1999);
- continuous product updates. Rapid changes in the product market place and user needs makes COTS evaluation difficult (Oberndorf 1999). For example, a new release of the product may have a feature that is not available in the product that is currently being evaluated, yet the user can demand that such functionality be included in their new system.

2.6.2 The problems of requirements in COTS software selection

Although, organisations experience different kinds of problems during COTS-based systems development process, most of these problems are due to inadequate requirements engineering and product evaluation and selection process, although they are rarely recognised as such. The requirements engineering process for the COTS-based development process is affected by problems that are very different from those of traditional systems development processes. The existing traditional development

life-cycle processes offer developers very little practical guidance to assist in the selection of specific products from the myriad available (Fox et al 1997). Furthermore, in traditional systems development, system requirements are defined in minute detail and then the system is built to the exact specification that matches those requirements. But in a COTS-based development, requirements need to be much more flexible and less specific (Thomas 1999, Place 1999). If requirements are too specific and inflexible it might be impossible to find COTS products in the market that adequately meet the requirements. Also, the notion of 'requirements' in COTS-based development is divided into what this thesis characterises as *Type A requirements* and *Type B requirements*. Type A requirements are characterised as the set of those requirements that the final system composed of COTS products must satisfy. Type B, on the other hand, are characterised as the independent set of requirements that govern the selection of each product that is included as part of the system.

2.6.3 Constraints on the COTS software selection process

In a COTS-based development process, the system is constrained not only by the constantly evolving requirements and market instability but also by the capabilities of the COTS products currently available in the market place. As figure 2.4 depicts, in a COTS-based development, requirements engineers should take great care to consider overlaps, dependencies and associations between products in the market, customer requirements and system architecture, since these all influence each other (Thomas 1999). Each of these should be considered simultaneously and iteratively and the system customers must accept the possibility that the resulting system might be a compromise among these concerns.

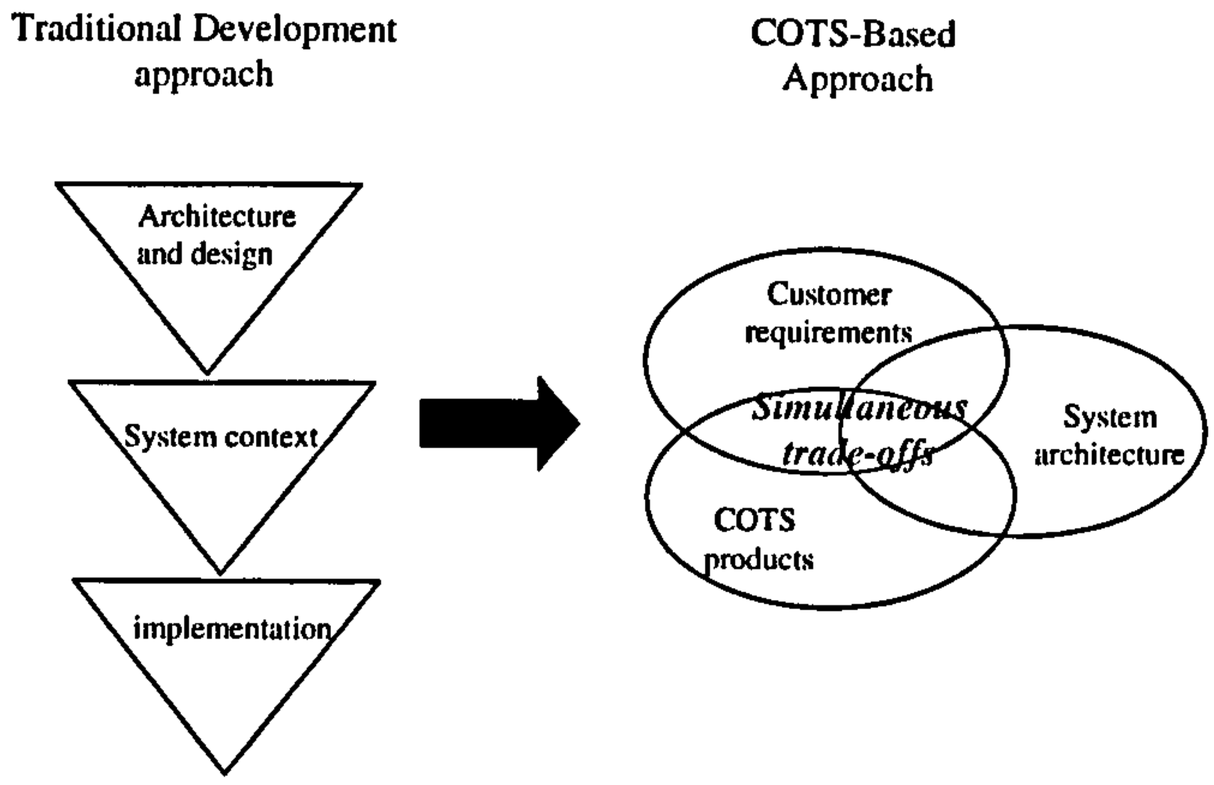


Figure 2.4: A COTS-based triad development approach (adapted from SEI 1999). Traditional systems development follows a strict sequence of activities. In the COTS-based paradigm there is a constant, simultaneous and iterative trade-off between acquiring customer requirements, evaluating and selecting products and the design of the system's architecture. In a traditional process, a system specification is produced up-front. In contrast during COTS-based development, the system specification may be the last activity to be performed after the requirements, products and their capabilities are known. The notion of development 'life-cycle' is also affected by the paradigm shift. Some activities such as contract production, evaluation, wrapping, bridging, have no analogy in the traditional life-cycle, yet they are of prime importance in the COTS-based development. Within this triad approach, the method described in this thesis does not explicitly deal with the design of the system's architecture. The method explicitly deal with customer requirements and COTS products.

In the traditional development process, the method of defining systems requirements is more straight forward: the desired system is described through a set of specified conditions that the system must meet. Requirements are fixed before building the system. However, defining requirements for the COTS-based development is very different since some systems requirements must be flexible enough so as to accommodate the fluctuations of the market place and the unforeseen constraints inherent in COTS-products. Since COTS products are usually developed with the software market place rather than the needs of specific developers in mind, requirements for the CBD process need to be more flexible and less specific (Place

1999). The consequences of narrowly specifying system requirements, as in traditional development, are that there might be no COTS products in the market that match those requirements. COTS product vendors usually design their products to meet requirements that they perceive to be the most likely to cover a wide market and make sales (Wallnau 1998). In COTS-based development, other factors such as market demands may determine which customer requirements are satisfied unlike in traditional development where requirements drive systems capabilities. Therefore defining requirements in great detail, as in traditional processes, establishes an artificial high baseline for evaluating and selecting the required COTS products (Vigder 1998). Too detailed requirements will lead to an unjustified elimination of candidate products that might otherwise provide reasonable system functionality (Dean & Vigder 1997). Therefore, in COTS-based development, initial requirements should be defined at a much more abstract level and detailed requirements should only be determined at much later stages of the process (Finkelstein et al 1997).

2.7 COTS software product evaluation and requirements engineering

Competing products are evaluated against requirements to determine products that sufficiently meet customer requirements. For this purpose, requirements must be defined in such way that will enable evaluation. Overly specified requirements can jeopardise evaluation and therefore prevent the selection of otherwise suitable products. Extensive evaluation of the COTS products is required not only to ensure that the product has the functionality to perform the required task within the system, but also to determine that the additional unwanted functionality inherent within the product will not interfere with the intended functionality of the system (Vigder et al 1996). However, most COTS-based development problems are largely due the unavoidable friction between customer requirements and the capabilities of the COTS software products. The activities of evaluating COTS products are closely tied to the activities of requirements acquisition and designing of the system architecture, as shown in figure 2.5. The evaluation activities usually span the entire lifetime of the system, i.e. they begin before the system is designed, continue as the system is being built, and even after it is deployed.

2.8 Problems with current COTS software evaluation processes

The importance of requirements engineering in the COTS product evaluation process is not reflected in the current state-of-the-practice. The present state-of-practice in COTS evaluation is poor (Oberndorf 1998), disconnected from the requirements acquisition process, and does not reflect the diversity of techniques and methods needed for product evaluation (SEI 1998). Some of the general misconceptions about COTS evaluation include:

- that evaluating a product is a one-time activity, with several products compared against a common set of usually weighted criteria (e.g Kontio 1996);
- that evaluation is a form of acceptance testing, where requirements are specified, and products evaluated for conformance with the requirements (e.g. Vidger et al 1996);
- that a common standard practice of evaluation can be defined and then re-applied for all COTS product evaluation session (e.g. Kontio 1996).

The influence of evaluation in requirements acquisition and architecture design is depicted in Figure 2.5. The figure contrasts the traditional development that does not include product evaluation and the COTS based approach in which product evaluation is the integral part of the development process.

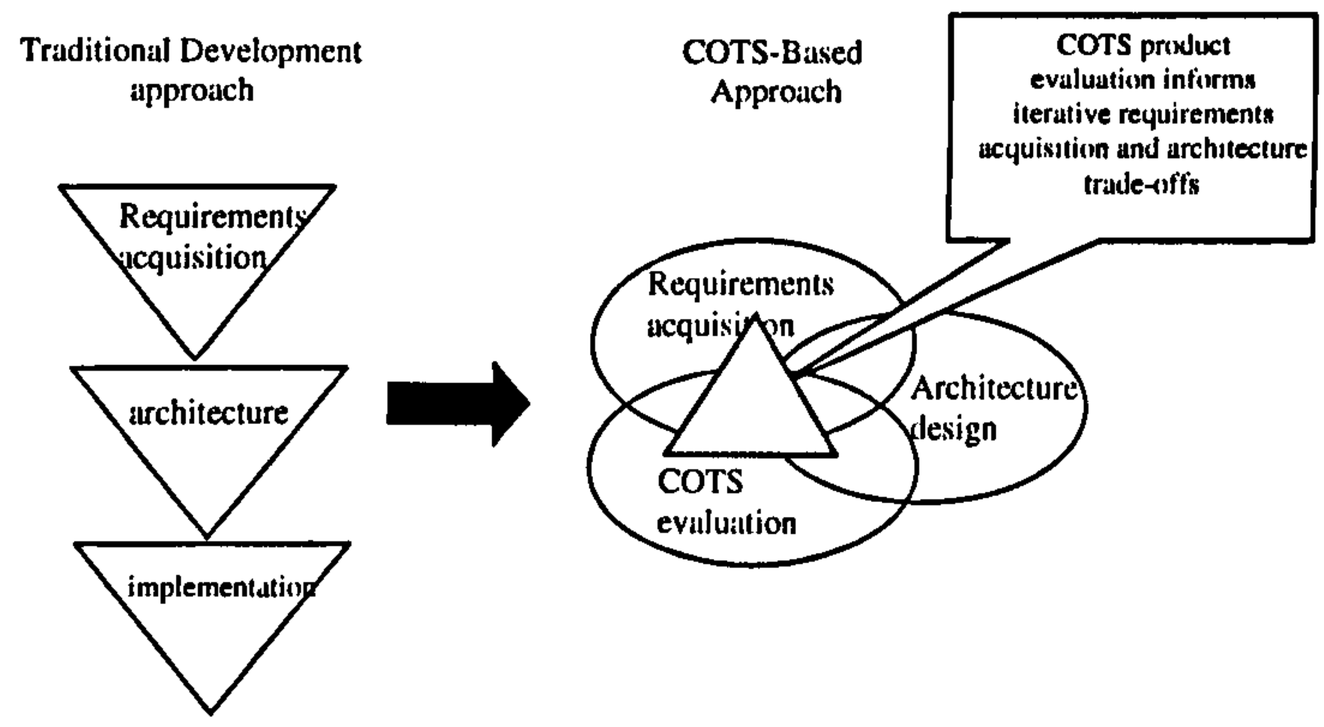


Figure 2.5: Influence of COTS evaluation on the requirements acquisition and architecture design processes (SEI 1998). This shows the contrast between the traditional approaches, which do not include the evaluation process, and the COTS based approach in which product evaluation is an integral part of the

development activities. In the traditional development processes, requirements are specified, then high level architecture design is produced, followed by detail design and ultimately implementation (i.e. waterfall design). In contrast, in the COTS-based development process, there is a constant simultaneous trade-off between requirements acquisition, product evaluation and architecture design.

Another mistaken assumption about COTS product evaluation is the idea that product evaluation is a one-off event for each selected product. Rather, there are evaluation activities that precede evaluation for the selected product and evaluation activities after the product has been selected (e.g. Carney 1998, Thomas 1999). Some evaluation activities are even performed concurrently (Dean 1999). This multi-stage evaluation process usually happens when there are several candidate products to be considered and where new versions of these products are emerging in the market sufficiently rapidly to justify deferring some aspects of the evaluation until more information is known. It is this multi-stage evaluation that is important to ensure that product evaluation does not depend upon perfect and complete product knowledge or customer requirements.

2.8.1 Emerging COTS software evaluation strategies

Two product evaluation strategies seem to be emerging from current practice (Tran & Liu 1997):

- *strategy 1* is where requirements for different parts of the system are acquired and product alternatives that implement these different parts of the system are evaluated against core essential functional, non-functional, system architecture and integration requirements;
- *strategy 2* is where alternative integration configurations are evaluated using core system architecture, interoperability and integration requirements.

However, these strategies have some undesirable side effects. The problem is they create dependence relationships among selection decisions and among products and customer requirements. However, the strategies help to partition requirements into two levels or sets – requirements for evaluating products that implement different parts of the system (i.e. *Type A requirements for strategy 1*) and requirements for evaluating the overall system composed of the selected products (i.e. *Type B requirements for strategy 2*). This realisation of different levels of requirements for product evaluation is one of the main reasons this thesis is advocating a process that is able to support iterative concurrent requirements acquisition and product evaluation. The iterative process provides a flexible link between product evaluation, requirements acquisition and system architecture design.

Tran & Liu (1997) also suggest three further important areas of COTS product evaluation:

- functionality evaluation that evaluates both the functionality of the individual products and integration of these products;
- interoperability and architecture evaluation which ensures that selected candidate products can be integrated according to their specification;
- performance evaluation which addresses the performance of the integrated products in supporting system level functionality.

Tran & Liu (1997) also identify two approaches for organizing the COTS product evaluation process – the *Comprehensive Evaluation* (CE) and the *First-Fit Evaluation* (FE), with other evaluation approaches falling in between these two extremes:

- in the Comprehensive Evaluation (CE) approach, all sets of the candidate products are evaluated through all identified stages. This results in a prioritised list of product sets ranked by their overall performance. The CE approach ensures that an optimal product set will be selected for the final integration at the cost of additional evaluation time and resources.

- the First-Fit Evaluation (FE) approach on the other hand, ensures minimal cost to the evaluation effort by eliminating product sets that failed in a

particular evaluation stage and selecting the first one that passes all the evaluation stages even though the selected product might not be the optimal solution.

However, the drawback of both the strategies and the approaches is that they fail to recognise the advantages of requirements-driven evaluation and the positive impact requirements have on the evaluation process. This failure to recognise the impact of requirements in COTS-based development is also evident in current methods being developed. The following section gives an overview of five methods. However, this is not a complete set of all currently available methods.

2.9 Current COTS-based systems development methods

A range of COTS-based development methods has been proposed. The following sections describe 5 such methods, and their limitations and weaknesses.

2.9.1 The Off-The-Shelf Option (OTSO) method

The Off-The-Shelf Option method (Kontio 1995) aims to address the problems associated with the selection process for off-the-shelf products. OTSO supports the search, evaluation and selection of COTS products. The method attempts to provide specific techniques for defining evaluation criteria, comparing the costs and benefits of alternative products, and consolidating the evaluation results for decision making. The OTSO evaluation criteria definition process decomposes requirements for COTS products into a hierarchical criteria set categorised into four groups:

- functional requirements for the COTS product;
- required product quality characteristics;
- business concerns such as cost, vendor stability, etc;
- relevant product architecture.

The main characteristics of the OTSO method are:

- a defined, systematic process that aims to cover the whole selection process;
- a systematic method for deriving detailed off-the-shelf (OTS) evaluation criteria from the system goals;
- a method for estimating the relative effort or cost-benefits of different alternatives;
- a method for comparing the 'non-financial' aspects of alternatives, including situations involving multiple criteria.

The OTSO method identifies five factors that primarily influence the selection of a COTS product:

- the application requirements such as functional and non-functional requirements. The requirements specification, is used as the basis for interpreting these requirements;
- the application domain and architecture requirements which may pose additional constraints for the evaluation. The application environment may rule out some incompatible alternatives; software architecture or design may make integration of some alternatives impractical and application domain may have some specific characteristics that are not addressed by COTS products developed for other domains;
- cost requirements – project objectives and constraints such as budget and schedule may influence the selection;
- availability of required (or not required) features in potential COTS candidates may affect the selection of the product;
- an organisation's system infrastructure and maturity should be considered when defining an evaluation process.

However, even though OTSO realises that the key problem in COTS selection is lack of attention to requirements, the method does not provide or suggest any solution. The main focus of the method is on defining the evaluation criteria but it does not offer guidance on how to acquire the application requirements against which to evaluate the products. The method assumes that requirements already exist since it

uses a requirements specification for interpreting the requirements. As already mentioned before, the idea of having a requirements specification implies that requirements are defined then signed off and frozen. However, this assumption is false in the COTS-based development paradigm.

2.9.2 The COTS-based Integrated Systems Development method, (CISD)

The COTS-based Integrated Systems Development (CISD) method (Tran & Liu 1997) aims to address various aspects of the CBSE development process. In particular, CISD addresses problems and costs associated with identification and integration of COTS products. CISD is a procurement-centric method that aims to provide an accurate reflection of the development steps associated with the implementation of component based integrated systems. CISD attempts to generalise the process of selecting, evaluating and integrating COTS products. The method consists of three key phases - *product identification, product evaluation and product integration:*

- the **product identification phase** includes the process of collecting and understanding overall system requirements, identifying and classifying COTS products into sets and prioritising them for subsequent evaluation. The major activities for this phase include: (1) the requirements analysis and classification stage that encompasses the process of understanding the system requirements and prioritising them into various application and service domains; (2) the product identification stage which collects information on candidate COTS products and groups them into different combinations or sets for further evaluation; (3) the product prioritisation stage that includes the review of all candidate product sets to generate a prioritised list for further evaluation.

- the **product evaluation phase** includes the process of evaluating and comparing the product sets to select the most optimal combination for integration. The goal of this phase is to compare and identify an optimal set of collaborative COTS products for the initial integrated system. The phase includes the following evaluations:

- evaluating the functionality of individual products and product sets, i.e. the actual verification and validation of the overall capabilities of individual products and sets of products;

- evaluating the architecture and interoperability of products, i.e. evaluating the connectivity and architecture of candidate products. Main issues addressed include: (a) the interactions of products along identified critical system paths; (b) the extensibility of the overall integrated architecture, and (c) the compliance with the required standards by the integrated products;

- evaluating the performance of individual products and product sets, i.e. performance evaluation of all interacting products to provide detailed understanding of their impact to the overall system's performance.

- the **product integration phase** includes building of necessary product adapters and enhancements to the selected product sets to implement the required system functionality.

Although the CISD method's product selection, evaluation and integration processes can be performed iteratively, it integrates development-centric approaches, such as the waterfall and spiral with procurement-centric approaches. The method is generally a waterfall-style process in that each stage depends on the results of the previous stage (Dean 1999) and does not offer solution on how to acquire customer requirements for COTS-based development. The CISD method heavily depends on having a complete predefined set of requirements. The product identification phase is dependent on the COTS product meeting these predefined requirements, and this is a drawback if the requirements are less than fully defined.

2.9.3 The Infrastructure Incremental Development Approach, (IIDA)

The Infrastructure Incremental Development Approach (IIDA) method (Fox et al. 1997) combines the classical development model and the spiral process model to accommodate COTS-based infrastructure development. Each stage of the development cycle is augmented with a series of prototypes for COTS product evaluation and integration. It is this close coupling of prototyping and development that characterises the IIDA method. Although IIDA method combines the waterfall and spiral model like the CISD described above, the key difference is that IIDA emphasises the establishment of product compatibility and completeness rather than product-level specification.

The IIDA also strongly emphasises testing and is heavily prototype-oriented. It provides a prototype-driven approach to COTS selection and integration and specifically focuses on addressing infrastructure development of large distributed information systems and not of applications. The method uses *Analysis Prototypes* to identify leading candidate products in each product family. Product families are defined as a group of products that perform similar functions and/or provide related services. *Design Prototypes* are used to exercise the product to determine its functional capabilities and how well it performs in accordance with its documentation.

The IIDA is a tailored lifecycle that preserves the benefits of existing structured processes for software development while adapting to the particular characteristics of integrating COTS products. The IIDA is a combination of the classical waterfall development model (Royce 1987) and the spiral development model (Boehm 1988). It is an iterative and incremental approach to infrastructure development where each version of the infrastructure is an increment that is integrated into the existing infrastructure baseline. With each version, development proceeds in time-sequenced stages with iterative feedback to preceding stages.

The main stages of IIDA are:

- **Definition and Analysis Stage** in which: (1) enterprise requirements and standards, system architecture and technical strategies are defined and

refined; (2) version-specific functional infrastructure requirements are established by considering business application areas, architectural imperatives and technology availability.

- **Functional Design Stage** in which: (1) services included in the target and current versions are identified and defined; (2) prototypes are used to identify leading candidate COTS products.

- **Physical Design Stage** in which: (1) interfaces between applications and infrastructure are defined (APIs are established); (2) COTS and to-be-built components are identified; (3) prototypes are used to select and characterise COTS components; (4) design is calibrated for scaling and performance considerations; (5) structure of each to-be-built component and its interfaces is defined.

- **Construction Stage** in which: (1) to-be-built components are constructed; (2) glue code is developed and the unit is tested; (3) COTS component, glue code, and built components are integrated into the infrastructure using the demonstration prototype as a test bed.

- **Test Stage** in which infrastructure versions are tested prior to being integrated and tested with business applications

Although IIDA includes COTS evaluation and integration in its development process, it is not clear how requirements are used in the evaluation process. The underlying assumptions of the method are heavily influenced by traditional development processes and therefore suffer from the same shortfalls and problems. Its main focus is on infrastructure development but not developing systems from COTS products. As with the COTS-based Integrated Systems Development Method (CISD) described above, IIDA is essentially a waterfall model with each stage dependent upon successful completion of the previous stage.

2.9.4 IusWare – A methodology for the evaluation and selection of software products

The IusWare method (Moriso & Tsoukias, 1997) is based on the Multicriteria Decision Aid (MCDA) approach and encompasses such activities as comparison assessment and selection of software products. The method defines an evaluation process that consists of two main phases – designing an evaluation model and applying it. The design phase activities include:

- identifying actors relevant to the evaluation, their role, the purpose and objectives of the evaluation and the resources available;
- identifying the type of the evaluation required – either a formal description of products or the ranking of products from the most preferred to the least preferred or a partitioning them into two sets of best and rest products;
- defining a non-redundant hierarchy of evaluation attributes, often corresponding to characteristics of quality models;
- associating a measure, a criterion scale and a function to transform the measure scale into the criterion scale for each basic attributes;
- choosing an aggregation technique for aggregating values on criteria for recommending selection;

In the application phase, attributes of products are measured and the measures are transformed into values on criteria and aggregated in a recommendation. In order to produce reliable recommendations, the method defines the following key points:

- designing of verification activities to check the independence of the product being evaluated;
- the aggregation technique is not considered to be a constant, but a key variable of the evaluation model, to be chosen consistently with the other components of the model;
- the methodology assumes that judgement is always present in an evaluation and the ultimate goal is to formalise its use and to merge measurement and judgement. For this the method identifies where a

judgement is involved and compels the actors participating in the evaluation process to openly declare and discuss subjective choices.

Although IusWare is a methodology for evaluating and selecting software products, it does not at any stage deal with the issue of requirements for product evaluation. The method presents an evaluation process model which involves defining a hierarchy of evaluation attributes together with a decision-making model that is based on the MCDA approach but assumes that requirements are already known. This is the fundamental weakness of the method.

2.9.5 Feature Analysis evaluation method

The Feature Analysis evaluation method (Kitchenham 1996, Kitchenham & Jones, 1997) attempts to put rationale and structure behind a 'gut feeling' for selecting the right product. The main important part of the feature analysis is:

- to help clarify the important features of the product in the context of the environment in which it will be used;
- to help identify the differences between the products; and
- to provide an explanation of why a decision was made to select it.

The features of the feature analysis method are:

- **Iterative Procedures** – feature analysis is a shortlisting activity that is performed iteratively. Each iteration reduces the number of candidate products and each iteration can vary the set of features being assessed, the individuals making the assessment, or the evaluation criteria. The starting point of each iteration and the number of iterations depend upon the number of products and the amount of effort available for performing the activity;
- **Comparative Framework** – the common framework is used for making comparative evaluations. The framework is expressed in terms of a set of common mandatory and/or desirable properties, qualities, attributes, characteristics or features for each type of product;

- **Subjective Judgement Scales** – products are judged by how much support they actually provide to achieve each customer requirement, i.e. to what degree does the product meet the requirement. The method devises a scale for assessing the degree of support that a product provides for a specific feature. Those products that possess a feature that score highly on the scale are judged to be "good" and those that do not score highly are judged to be "less good";
- **Product Assessment** – once each product has been "scored" for each feature in the framework using some common judgement scale, the results are compared to decide their relative order of merit;
- **Feature Complexity** – features chosen for evaluation may be very complex. The method decomposes them into sub-features that are conceptually simpler and the sub-features are further decomposed. However, there is no rule that says when to stop decomposing, therefore it is very easy to generate a large number of features.

Feature analysis is based on identifying the requirements that users have for a particular task/activity and mapping those requirements to features that a product aimed at supporting that task/activity should possess. Products are compared feature by feature. Evaluators assess how well the identified features are provided by a number of alternative products. However, the method does not offer any solutions as to how to address the requirements acquisition problems. Also, the method does have several significant drawbacks and limitations. Its main limitations are:

- **Subjectivity** – feature analysis is based on judging products against some "evaluation criteria" that are identified subjectively. Such evaluations are likely to be biased and context dependent;
- **Inconsistency** – there is also a problem of inconsistency in scoring between different assessors. If different assessors evaluate different products, they may have different degrees of familiarity with and understanding of the product. In addition, they may interpret the scale for a particular feature in different ways;

- **Collating Scores** – producing a set of scores for all the features for a specific product is, unfortunately, not the end of the story. The various scores have to be collated and compared to decide the relative order of merit of the methods or products and this could result in a biased final score.

2.9.6 Summary of the current COTS-based methods

Almost all the methods discussed above do not adequately deal with the problems of requirements acquisition. Those that recognise the problem of requirements do not offer or propose any solutions. Also, most of these current COTS-based development methods have processes that identify strict requirements which either exclude some COTS products or that will require large modifications to the products in order to satisfy the requirements. Table 2.1 below shows the main COTS-based development processes that are covered by the current methods. The table shows that almost all the currently available methods have an overall weakness in dealing with requirements engineering. None of the methods adequately address the issue of requirements acquisition, while almost all of them focus on the product evaluation/selection process.

	Product Identification	Requirements Acquisition	Evaluation/ Selection	Integration	Design and Development	Testing	Decision Analysis
OTSO	√	–	√	–	–	–	√
CISD	√	–	√	√	–	–	–
IIDA	–	–	√	√	√	√	–
IusWare	–	–	*	–	–	–	√
Feature Analysis	–	*	√	–	–	–	*

√ addresses the issue fully, – does not deal with the issue, * deals with issue but not fully

Table 2.1 Summary of the processes that are covered by the current COTS-based development methods. Table 2.1 shows that almost all current COTS-based development methods do not address requirements acquisition. Most of these methods address evaluation issues.

2.10 Decision-Making techniques.

Another critical issue for the COTS-based development process is the issue of product assessment and decision-making. In order to select or recommend a suitable required COTS product, the evaluated alternatives must be ranked according to their perceived relative importance to meet the customer's requirements. Decision-making techniques have been used for this purpose.

Making a decision that does not help to achieve the goal of selecting the required product can lead to long lasting user disappointments. Decision-making in product evaluation is a very complex process that combines probability judgements that may be affected by the evaluator's beliefs and underlying preferences. Figure 2.7 depicts the principles of decision-making in product evaluation and selection that the users usually have to contend with, represented as a hierarchy of three levels taken from Saaty (1990). At the first level is the main goal for the decision making process (e.g. selecting a suitable product among the alternatives). At the second level there are some criteria for selecting the product. The suitable product will be judged by these criteria. At the third level are the actual alternative candidate products in which the criteria will be applied to achieve the main goal.

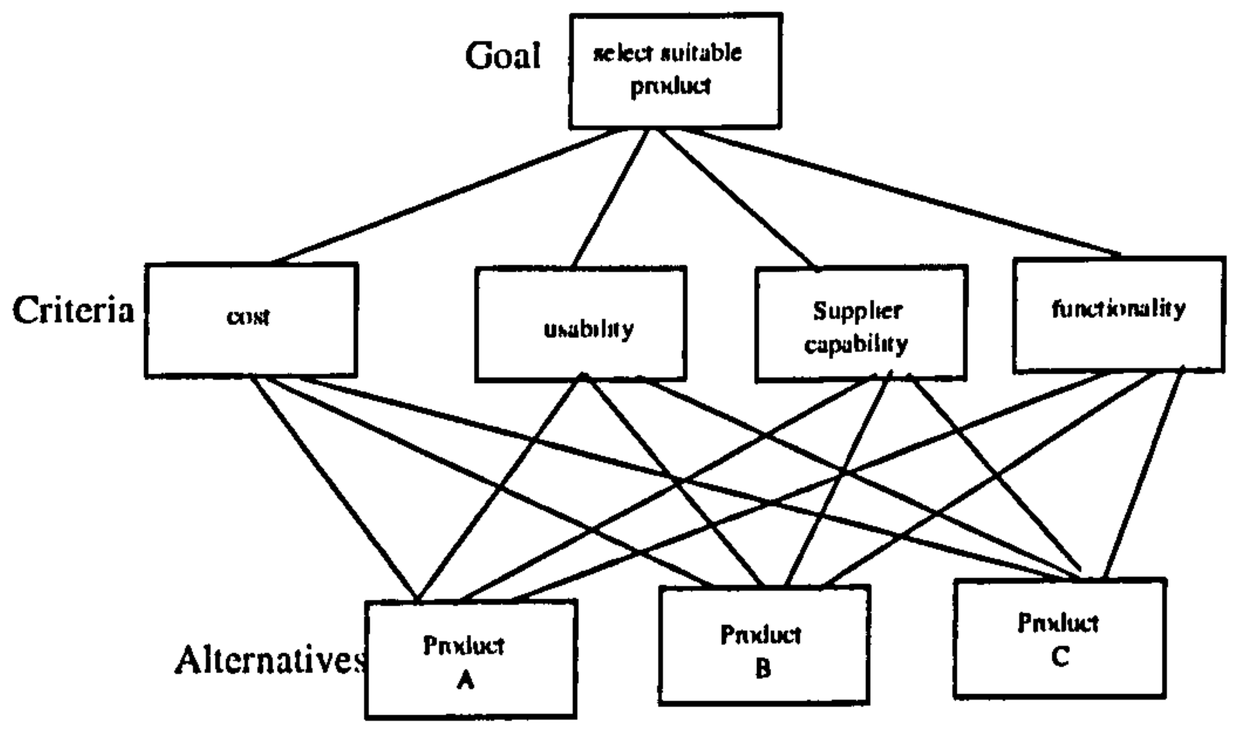


Figure 2.6 Principles of a decision-making problem. The figure clearly shows how complex decision-making becomes if there are many products to compare and criteria to apply to each product.

2.10.1 General difficulties with current decision-making techniques

Anderson (1989) describes factors that give rise to problems in evaluating and assessing COTS products:

- that there is a large number of product attributes or features that have to be considered;
- that various combinations of hardware platforms, operating systems and application software need to be considered;
- that there is rapid technological changes in all aspects of computing, the business environment and the needs of the users;
- that most users lack the technical expertise or time to develop criteria, measurements and testing procedures for performance assessments and to conduct the actual evaluations;
- that there are considerable variation in performance between the attributes of each product and across the products for each attribute.

Currently, a number of decision-making techniques that can be used in COTS product evaluation and assessment are available in the market. However, almost all of these techniques are not suitable for assessing software products due to their fundamental underlying assumptions in their judgement value system. Most currently existing traditional decision-making approaches rely on compensatory models such as the linear weighted score model which sums the weighted ratings of the product's capability attributes to arrive at a single score for each product. These models incorporate some means of scaling and weighting the importance of various product attributes to give a mathematical means of combining magnitude and significance for the overall evaluation of the product. Scaling addresses issues of magnitude and is based on a numerical system in which the highest number represents a very good score and the lowest number represent a lower score with the mid-point representing an average score. With weights, values are given to the product's attributes based on their relative importance or significance to the user. The end result is either an aggregate total score for the product or a group of scores representing various attributes of the product. However, aggregate total scores tend to mask individual attributes of the product that may represent particular strengths or weaknesses in a

product. These models are problematic in that they can permit very good performance on one attribute to offset poor performance on another (Anderson 1989).

Definitions of all criteria are required so that all COTS alternatives can be compared against a common yardstick. Anderson (1989) suggest three factors or conditions in which comparative conclusions about the quality of the *ith* and *jth* products can be drawn:

- the frequency with which the attribute performance of the *ith* product, weighted by attribute importance, exceeds that of *jth* product, (e.g. outranking);
- the extent to which there is substantial differences in quality between the two products on the one attribute that could be glossed over by aggregates or averages;
- from the sheer magnitude of differences between the attribute ratings.

Once the evaluation of alternative products has been done, a common approach for consolidating the evaluation results and for ranking the alternatives is needed. The techniques summarised below all attempt to consolidate the evaluation results or rank the alternatives in one way or another.

2.10.2 Current proposed decision-making techniques

Various decision-making techniques have been proposed. The following sections 2.10.2.1 – 2.10.2.5 give brief discussions of five such decision-making techniques.

2.10.2.1 The Multi-Attribute Utility Theory (MAUT)

The MAUT decision-making model deals with choosing among a set of alternatives which are described in terms of their attributes (MacCrimmon, 1972). A typical multi-attribute decision problem is choosing among software products described by such attributes as cost, usability, functionality, size, portability, supplier capability etc. To deal with the multi-attribute situations, the MAUT technique requires information about:

- the decision-makers preference among values of a given attribute (i.e. how much does s/he prefer a commercial database over a proprietary database), and;
- the decision-maker's preference across attributes (i.e. how much important is the database than cost). A marginal value function is associated with each criterion and a global value function is computed in an additive or multiplicative form.

The MAUT technique asks the decision-maker for an assessment on the strength of preferences. The decisions may the be reduced to a number of independent attributes that involve making trade-offs between different goals or criteria. MAUT uses a reductionist approach to a problem and it is up to the decision-maker to split the problem into a number of dimensions that are perceived to be independent. This independence is essential for MAUT because without it, certain attributes could be over represented in the final result. This is the fundamental weakness of the method.

2.10.2.2 The Multi-Criteria Decision Aid (MCDA)

The MCDA approach is in the category of the utility theory. Morisio et al (1997) proposes the following advantages for using MCDA in COTS product evaluation and selection:

- the MCDA approach makes explicit reference to the decision process so as to take into account the different actors involved in the process, their different objectives and the partiality of the information provided;
- the MCDA approach allows to handle judgements based on qualitative, partial information and the subjective nature of the problem of evaluating and selecting software products. This is done by adopting appropriate specific techniques to help in the decision making process (including multi-attribute utility theory, multi-objective interactive techniques and out-ranking technique) and provide the evaluator with both formal and substantial reasons for any choice;
- the MCDA approach combines strictness (non-redundancy of the set of criteria, appropriateness of the aggregation procedure, etc) with flexibility,

different problem statements, different aggregation techniques, custom evaluation attributes and measures.

With MCDA, a list of criteria that the product should meet is established first, then scores are assigned to each criterion based on its relative importance in the decision. Each alternative is then given a number of scores according to how it fully meets the criterion. For the scores, a scale of 1 to 5, or 1 to 7, etc can be used. An example is shown in Table 2.2. The main weakness of the method is that if the criteria set is large, it quickly becomes very complicated.

Criteria	Possible Points	Product A	Product B	Product C
Cost	30	25	20	15
Functionality	40	35	10	20
Supplier	20	15	5	10
Usability	10	5	3	2
Total	100	80	38	47

Table 2.2 A list of criteria that the product should meet and scores assigned to each criterion. In the example, product A is rated 25 out of 30 points for the "cost" criterion, while product C is rated a little less favourable. Once all the alternatives have been assigned their points for each criterion, all points for each alternative are added together and the alternative with the highest total is the one chosen. In the example above, this would be product A.

2.10.2.3 Weighted Score Method (WSM) or Weighted Average Sum (WAS)

The WSM/WAS is an aggregation technique and the most commonly used technique in many decision-making situations. Its 'weights' are trade-offs between the criteria, i.e. they are ratios between the scales of each criterion. *'Criteria are defined and each criterion is assigned a weight or a score'* (Kontio, 1996).

The scales themselves represent preferences relative to each attribute. The WSM/WAS technique is a fully compensatory model in that each preference relative to a criterion can be totally compensated for by a countervailing preference on another

criterion. This trade-off between criteria may result in any big difference that may exist being compensated for, so that an indifferent situation is created instead of the actual incomparability situation (Morisio et al 1997). This scenario is one of the many weaknesses of this technique. Although weighting methods seem very diverse, they all have the following characteristics:

- a set of available alternatives with specified attributes and attribute values;
- a process for comparing attributes by obtaining numerical scalings of attribute values (intra-attribute preferences) and numerical weights across attributes (inter-attribute preferences);
- an objective function for aggregating the preferences into a single number for each alternative;
- a rule for choosing or rating the alternatives on the basis of the highest weight.

Table 2.3 below shows an example application of the WSM and its limitations

Criteria	Weight Score	Product A	Product B	Product C
Ease of use	2	3	3	3
Compatibility	4	1	5	2
Cost	3	3	5	1
Functionality	5	4	4	3
Security	4	1	2	5
Supplier	5	2	5	3
Score		53	94	67

Table 2.3 Example of the weighted score method. The criteria weights were assigned using a scoring method by assigning a value of between 1 and 5 to each criterion. The overall score of each alternative was calculated using the following formula (Kontio, 1996):

$$score_a = \sum_{j=1}^{n} (weight_j * score_{aj})$$

where a = alternative, n = number of criteria, j = criteria

The problem with the method is in assigning the scores. For example, the security and compatibility could be interpreted as twice as important as ease of use, whereas in reality this might not be the case.

However, WSM/WAS techniques have serious limitations that are often ignored when they are applied in COTS product evaluation and assessment (Kontio 1996):

- As the Weighted Score Method produces real numbers as results, these results can easily be interpreted as if they represent the true differences between the alternatives. In actual fact, the resulting scores only represent relative ranking of the alternatives and the differences in their value does not give any indication of their relative superiority;
- Assigning weights for the criteria is very difficult when the number of criteria is large. If the number of attributes is large, it is very difficult to mentally cope with the dependencies between individual attributes. Assigning scores instead of weights is even more limiting because it effectively sets predetermined lower and upper limits to the weights that can be assigned to the criteria;
- It is very difficult to define a set of criteria and their weights so that they are either independent from each other or if they overlap, their weights are adjusted to compensate for the overlap.

2.10.2.4 The Analytical Hierarchy Process (AHP)

The AHP (Saaty 1990) is a multiple criteria decision-making technique that is based on the idea of decomposing a multiple criteria decision-making problem into a hierarchy. The decisional goal is decomposed into a hierarchy of goals and ratio comparisons are performed on a fixed ratio scale. The overall priorities are computed using an eigenvalue technique on the comparison matrix. The factors are arranged in a hierarchic structure descending from an overall goal to criteria, sub-criteria and alternatives in successive levels as shown in figure 2.6. At each level of the hierarchy, the relative importance of each product attribute is assessed by comparing them in pairs. The rankings obtained through the paired comparisons between the alternatives are converted to normalised rankings using the eigenvalue method, i.e. the relative rankings of alternatives are presented in ratio scale values which total to one as shown in the Priority Vector column of Table 2.4. The technique suggests that comparing

criteria in pairs result in more reliable comparison results and that in this way, it is possible to avoid the problem of having to assign absolute values to alternatives, but only their relative preferences or values are compared. A typical application of the AHP method is shown in Table 2.4. Functionality is shown to have the highest total score and priority vector and therefore ranked more important.

			Level 1 Priority Vector				
	Cost	Functionality	Usability	Technical	Supplier	Total Scores	Priority Vector
Cost	1	4	5	4	6	20	0.339
Functionality	0.25	1	7	7	7	22.25	0.377
Usability	0.2	0.143	1	5	3	9.343	0.158
Technical	0.25	0.143	0.2	1	4	5.593	0.095
Supplier	0.167	0.143	0.333	0.25	1	1.893	0.032
						59.079	1

Table 2.4: An example of applying the AHP method.

Frair (1995) proposes the following four steps in applying the AHP method:

(1) Build a decision hierarchy by breaking the general problem into individual criteria;

(2) Gather rational data for the decision criteria and alternatives and encode using the AHP rational scale (i.e. user pairwise comparison input);

(3) Estimate the relative priorities (weights) of the decision criteria and alternatives (i.e. either using the AHP software tool or a spreadsheet);

(4) Perform a composition of priorities for the criteria which gives the rank of the alternatives relative to the top most objective (i.e. in table 2.4, functionality is ranked highest).

However, the AHP technique has some fundamental drawbacks when applied to COTS product evaluation. One of its main problems is that it assumes total independence between the product attributes, i.e. in order to do a pair-wise comparison, the technique assumes that the product attributes/features are independent of each other and this is rarely the case with software requirements. Also, especially for large complex systems, it is difficult to apply the AHP technique as its

calculation model involves a very high number of pair-wise comparisons. The large number of individual assessments is also one of its main weaknesses. Even if the overall duration of the assessment sessions are not very long, the repetitive assessments cause tiredness and boredom. Furthermore, the assumption that there should be complete comparability and the imposition of the ratio scales at all levels of the hierarchy is very demanding (Kontio 1996).

2.10.2.5 The Outranking method

In the outranking method (Fenton 1994) a global preference relation is computed via direct aggregation of the preference structure and then exploited to compute the prescription. There are many aggregation and exploiting procedures that enable the evaluator to tune the technique to the problem situation. Unlike other techniques, the outranking technique distinguishes between classification and choice and solves the problem of sorting factors. Outranking methods '*seek to enrich the dominance relation between products without having to make the strong assumptions necessary for other MAUT methods*' (Fenton 1994). An outranking relation is defined by Fenton (1994) as 'a binary relation S on A such that aSb if, given what is known about the decision maker's preferences and given the quality of the valuations of the actions and the nature of the problem, there are enough arguments to decide that *a is at least as good as b*, while there is no essential reason to refute that statement'. The outranking relation is not necessarily complete or transitive, i.e. aSb ≠ bSa. The outranking method has two main steps:

(1) build the outranking relation
(2) exploit the relation with regard to the chosen problem statement

The following example taken from Fenton (1994) illustrates how to build the outranking relation and therefore how the method works:

Let {g1, g2, g3, g4} be set of criteria as shown in Table 2.5. Each criteria maps actions into some ordered set. The seven actions represent different combinations of validation and verification techniques. For example action 1 might represent the

combination formal proof and code inspection, while action 2 might represent the combination formal proof and static analysis.

Action	g1: effort required	g2:potential for detecting critical faults	g3:coveraged achieved	g4:tool support
1	Excessive	Excellent	Good	Yes
2	Considerable	Excellent	Average	Yes
3	Considerable	Good	Good	Yes
4	Moderate	Good	Good	No
5	Moderate	Good	Average	Yes
6	Moderate	Reasonable	Good	Yes
7	Little	Reasonable	Average	No
Weight	5	4	3	3

Table 2.5 Criteria for assessing combined V&V Techniques. Taken from Fenton (1994).

The first thing to do in applying the outranking method is to assign a weight p_i to action g_i. In the table above, *effort required* is assigned a weight of 5 and *tool support* is assigned weight of 3. Next, for each ordered pair of actions (a,b) the weights of all those criteria g_j for which $g_j(a) \geq g_j(b)$ are added. For example, for the pair (1,2) in Table 2.5, action 1 is at least as good as 2 with respect to the criteria g2, g3, and g4. The sum of the weights is 10. This is called the *concordance index* and is written $c(a,b)$[2]. Morisio et al (1996) also define the *concordance index* as '*the majority strength to be reached in order to be able to establish the outranking relationship with a certain degree of confidence that is calculated using the relative importance of each criterion*'. The idea is that a is preferred to b if $c(a,b) > c(b,a)$. The preference structure is restricted by defining a *concordance threshold* t. The idea is that for a to be preferred to b we must have $c(a,b) \geq t$ and t should be set sufficiently large. In the example of Table 2.5, if $t = 12$, then action 2 is preferred to action 1. Next, the preference structure is refined to take into account of types of situations such as, for criterion g1 (effort required), it should never be allowed that an action a to outrank action b if **g1(a)** = *excessive* and **g1(b)** = *little*. This means that irrespective of the values of other criteria, b is so 'superior' to a with respect to $g1$ that a veto is put on it being outranked by a and this called a *discordance*. Generally, a *discordance* set $\mathbf{D_j}$ is defined for each criterion g_j. This is an ordered set of pairs (x,y) such that if $g_j(a) = x$ and $g_j(b) = y$, then the outranking of b by a is refused. Marisio et al (1996) further defines the *discordance* as the '*minority strength not to be reached in order to be able to establish the outranking relation that is computed using the relative importance of*

each criterion'. Once the *concordance threshold* and the *discordance sets* have been defined, the outranking relation ***S*** is defined as ***aSb***, i.e. action ***a*** outranks ***b*** provided that:

(1) $c(a,b) \geq t$, and

(2) for each criterion g_j, $(g_j(a), g_j(b)) \notin D_j$ holds.

2.10.3 Limitations of the current decision-making techniques

The current decision-making techniques suffer from similar problems as those experienced in current COTS-based development methods. The link between the current decision-making techniques and the current COTS-based development methods is shown in Table 2.6. Two methods do not use decision analysis in their development process while the OTSO method uses three techniques in various degrees.

	OTSO	CISD	IIDA	IusWare	Feature Analysis
MAUT	–	–	–	–	–
MCDA	√	–	–	√	–
AHP	√	–	–	–	–
WSM/WAS	√	–	–	–	√
Outranking	–	–	–	√	–

Table 2.6: The link between current COTS-development methods and decision-making techniques. Some methods do not have a decision-analysis process while those that have, the process is not adequately defined.

Therefore the selection of a decision-making technique for COTS evaluation and assessment should be done with care. This thesis suggests that a requirements-driven evaluation approach will have a positive impact on the evaluation and assessment process. Current decision-making techniques do not adopt a requirement-driven approach to product selection decisions and are therefore inadequate and not suitable for the COTS based decision-process. The fundamental criticism of these techniques is their underlying theory and their value judgement system, i.e. where the values come from.

For example, the idea of producing a single number from the individual scores (e.g. by some arithmetic combination formula such as weighted ranking) is misleading because many different combinations of numbers can produce the same aggregate score. Furthermore, certain features may attract higher average scores than others because an assessor may understand them better and be more able to recognise support in the product. There are also deeper reasons concerning the nature of the ordinal scales that are usually used to assess product features. For instance, a score of 4 is not necessarily twice as good as a score of 2.

As an example, suppose that a four criteria model for software quality is adopted using the following weights given by the client: cost (weight = 0.3), functionality (weight = 0.4), supplier capability (weight = 0.2) and usability (weight = 0.1), as shown figure 2.6 and in Table 2.2. Then when the WSM/WAS technique is used, the significance of these weights is that a unit of *functionality* is twice the unit of *supplier capability*, which is twice the unit of *usability*, etc. This is however, not necessarily correct since the immediate consequence is that a preference of four units in usability can completely compensate for an inverse preference of one unit of functionality. What is more, if the evaluations of the four criteria are not measures but arbitrary values which simply represent an order of alternatives, then there is an obvious contradiction between the nature of the information and the aggregation principle of the WSM/WAS technique which is unacceptable.

Therefore, the concluding view of this thesis is that most of the current decision-making techniques available are not adequate for COTS-based evaluations and assessments due to their underlying assumptions and their judgement value systems. There is a need for new requirements-driven decision-making techniques for the COTS-based development paradigm.

2.11 Summary and chapter conclusions

As organisations are increasingly shifting the development of their systems away from bespoke development to COTS-based development, this chapter has highlighted the current trends in COTS-based systems development research activities. This chapter concludes that there is a surprising lack of requirements engineering for COTS-based development. There is very little interest in the intersection of requirements engineering and product selection, in spite of the greater use of COTS products. Requirements are a cornerstone of effective COTS-based development. For example, requirements become criteria for product evaluation and selection; they are embedded in the legal contract. Requirements even provide acceptance criteria to check that the product being purchased and the system developed from the products will meet the customer requirements. When building systems from COTS products, the consequence of inadequate requirements acquisition and product evaluation is large because requirements changes can often incur great additional costs to the customer. However, despite the importance of requirements engineering to COTS-based development, this is not reflected in the current range and focus of commercial techniques, methods, tools and research efforts.

Research into requirements engineering for the COTS-based development process is also rare. One exception is Potts (1995) who identifies the need for requirements engineering for off-the-shelf software products. Finkelstein et al (1997) provides a review of important factors and research ideas for procuring COTS products but does not offer any concrete solutions. Tepandi (1995) identifies numerous factors that influence COTS product procurement practices but also does not provide any guidance or prescriptive processes. Most current COTS-based development methods such as that of Kontio (1996), Tran & Liu (1997), Fox et al (1997) Morisio & Tsoukias (1997) and Kitchenham & Jones (1997) and research such as of Garlan et al (1995), Brown et al (1995) and Vidger et al (1996) support systems design and integration, but neglect the requirements acquisition and product evaluation and selection processes which precede design and integration. These methods and efforts provide very little practical guidance to developers to achieve the advantages of COTS software or to assist in the requirements acquisition process for selecting specific products.

There is a need for process definition for COTS products usage, and new lifecycle models for COTS requirements acquisition, evaluation, selection and integration. Requirements acquisition for COTS product evaluation and selection is complicated by the intrinsic nature of COTS software special characteristics such as complexity, incompatibility, inflexibility and transience, i.e. periodic updates (Fox et al 1998). Product updates often add new functionality that is not compatible with the other system components. On the other hand, remaining with older versions of the COTS product may cause future interoperability problems with upgrades to other products.

Therefore, new methods and techniques for requirements acquisition and product selection and for guiding the COTS-based systems development process are needed. This thesis proposes a new method, PORE, (Procurement-Oriented-Requirements Engineering) to address the lack of requirements engineering methods for COTS-based development. The method supports and guides the requirements acquisition and product evaluation and selection processes for COTS-based development paradigm. PORE uses an iterative process of requirements acquisition and product evaluation/selection as its main novel approach and integrates existing requirements engineering techniques with those from several other disciplines. PORE is designed in part, from conclusions drawn from real-world case studies of requirements acquisition for complex software product selection.

The following chapters describe the research and case studies that contributed to development of the PORE method for COTS–based development. The following chapter reports on how data about current COTS software procurement and problems was gathered.

Chapter 3

Requirements Engineering Process and Method for COTS-Based Development

This chapter describes studies undertaken in procuring COTS-software Products and presents a case for the need of new methods and process guidance.

Chapter 3

Requirements Engineering Process and Method for COTS-Based Development

Two studies were carried out to investigate hypothesis 1:

H1 New problems arise from COTS-Based Development and procurement-oriented requirements engineering that are not addressed in current requirements engineering research.

The first was a study of three organisations which was carried out to inform the author as to how COTS-based systems are developed. Previous work had indicated that there is general lack of methods for developing systems from COTS software. Therefore an introductory study of 3 organisations' procurement processes was undertaken to gain more comprehensive knowledge about software procurement. The study investigated all stages of the procurement processes focusing on the selection of COTS software products and the stakeholders' perception of the selection process. Studies on the procurement of bespoke systems had been carried out, for example Tepandi (1995), Kemp (1995) and Potts (1996). These studies provided useful knowledge on methods, techniques and problems faced in the procurement of bespoke systems but not COTS software selection.

In addition, little knowledge existed about requirements for selecting COTS products and predominant requirements acquisition and product selection problems. Previous studies on COTS-based development tended to focus on systems integration, evaluation, design and architectures (e.g. Vigder et al 1996; Brown et al 1995; Garlan 1995) rather than on methods for acquiring requirements for product selection as depicted in figure 3.1. Therefore, a second substantive study was undertaken of the selection of a COTS requirements management system for the UK MoD. The study helped the author to gain knowledge about requirements acquisition for COTS software product selection and common problems that arise. The problems and

lessons learned during the studies informed the design of a first prototype method for requirements acquisition and COTS software product selection.

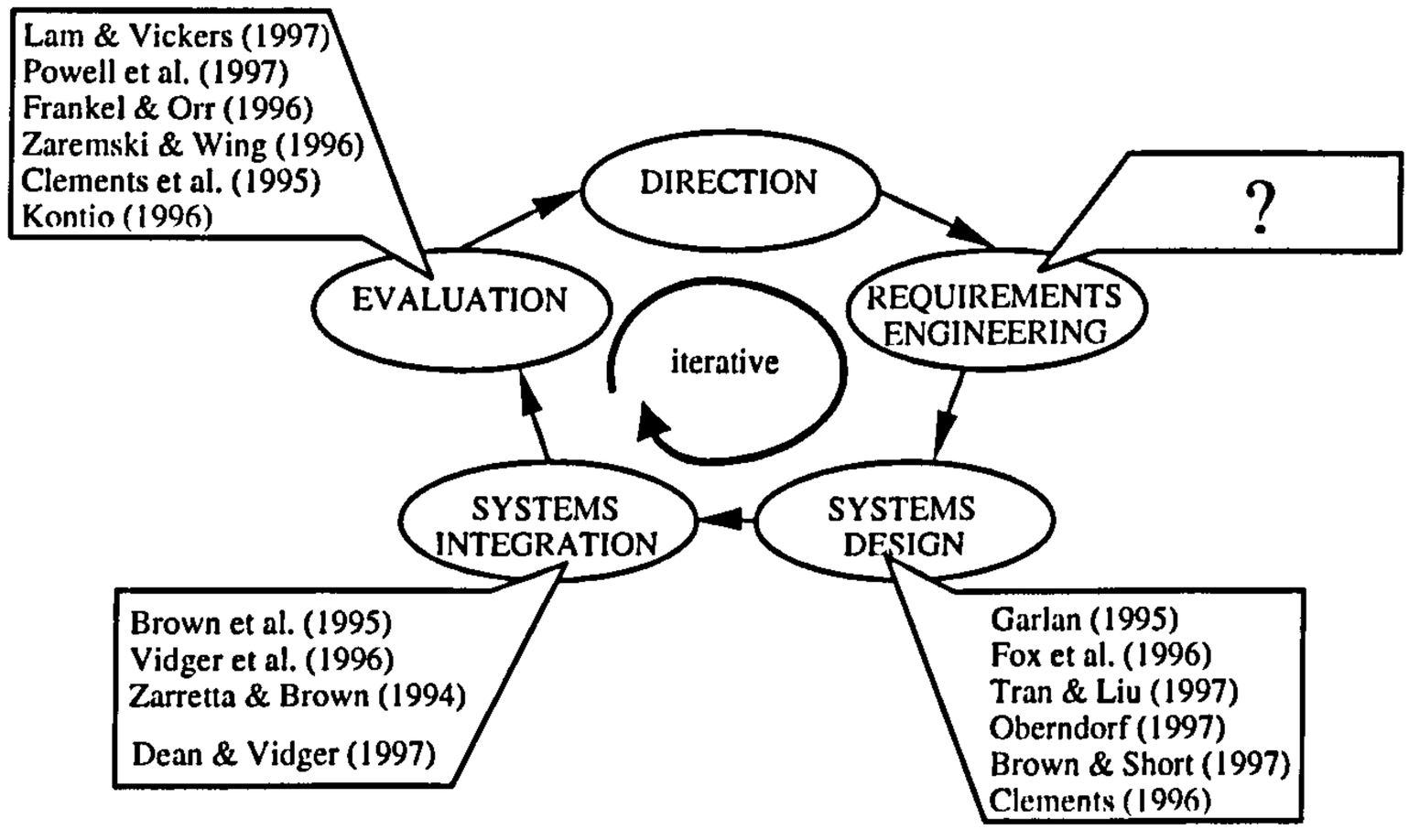


Figure 3.1: Most work in COTS-development focuses on systems design, systems integration and evaluation but none on requirements engineering.

3.1 Procurement process problems

3.1.1 The study method

The first study involved conducting 3 interviews with experts from 3 organisations. The first organisation (Organisation A) was a large defence organisation responsible for the procurement of large naval platforms and their software systems. These platforms have both information systems handling large amounts of data and real-time systems with critical performance requirements seldom known to commercial suppliers. A senior procurement executive described his task, responsibilities and problems. The second organisation (Organisation B) was an international airline. Its information management division provides software systems for all activities within the organisation including crew scheduling, rostering and flight planning. A cross-functional team with experiences in systems operations and information management recounted their experiences from the recent procurement of an interactive voice response system to improve cabin crew scheduling. The third organisation (Organisation C) was a small consulting organisation with over 20 years of experience

in requirements definition for systems procurement in both the private and public sectors.

3.1.2 Organisations' current procurement processes

Organisation A – organisation A is a defence procurement agency responsible for generating high level operational requirements for all types of ships, submarines and naval aircraft and their weapons, sensors and communications. The organisation is also responsible for generating the business case to achieve the endorsement of the operational requirements and the release of funds from the UK treasury. Its main task is to generate a comprehensive, coherent requirement specification for any of the navy platforms that covers the issues from design and build, and through service to disposal.

The organisation's procurement process start from the assumption that at some point in the future, a gap in the capabilities needed by the armed forces to meet the task laid down by the government has been identified. The current procurement process is based on what is called the Downey Cycle which came from a report done in the mid-60's (1966) for the RAF procurement division. The Downey study produced a model which says how a procurement should be done. The Downey Cycle is designed to take progressive stages to provide accountability in greater detail at one stage so that the organisation does not commit large sums of money until they can justify what they are going to get. The Downey Cycle divides a 10 – 15 year military procurement programme down into sizable chunks so that firm control on technology, programme and financial risks is retained.

Organisation B – the procurement process of organisation B starts by the Operations Division raising the business case which is then passed on to the Information Management Group (IMG). The IMG then conducts a project review to review the technical feasibility of the project. If the project passes the review it is then passed to another IMG steering group known as the Investment Review Group (IRG) who look at the justification for the project. If the IRG are satisfied with the project, the IT director signs the project and passes it to the operations director who is the end users.

Only after this stage can the users begin the process of procuring the right COTS software product.

Organisation C – organisation C is a small consulting house with over 20 years of experience in requirements engineering for systems procurement both in the public and private sector. Two interviews were conducted with the lead consultant. The consultant recounted his experiences and problems both in requirements definition for procuring software systems and the problems with the procurement processes. The lead consultant discussed the major causes of the problems at the all process levels in many organisations both in the public and private sectors. The following section describes how data was gathered from the three organisations.

3.1.3 Data gathering method

Data about procurement processes and problems was gathered from both documents and participants in the processes. Documents were examined for work flows, information use and business processes. Examples of documents examined include IT design documents from Organisation-B and current procurement procedures from Organisation-A. Interviews were conducted with groups and individuals in all 3 organisations. Some individuals were interviewed twice. During unstructured interviews participants were asked more open-ended questions about procurement processes and problems. Structured interviews asked more specific questions. At the end of each interview participants were shown the author's current version of the process model and asked to comment. All interviews were tape-recorded. In total 21 hours of interviews were undertaken. The following section describes the results of the case study.

3.1.4 Results

The data gathered from organisations was synthesised to produce the organisations' current procurement processes. Figure 3.2 shows the current procurement processes of organisation A. At the highest level, the procurement process is sequential but at the lower levels there are sub-processes and feedback loops. These sub-processes and feedback loops only appear sequentially if they have to, i.e. only if the output of

process one is required as input to process two. The main procurement processes are the *concept study, feasibility study, contracting, building and management, acceptance, service, mid-life modifications* and *disposal.* All the processes access a requirements database which once created will be used and needed for the next 30 – 60 years.

The problem with organisation A's current process is that it is a sequential process due to the way the UK treasury is involved: "*basically one thing is done and then stop, go get the treasury for authority to go the next stage and then start again*". However what is needed is a concurrent process so that some parts of the process could be going on while others are waiting for treasury approval.

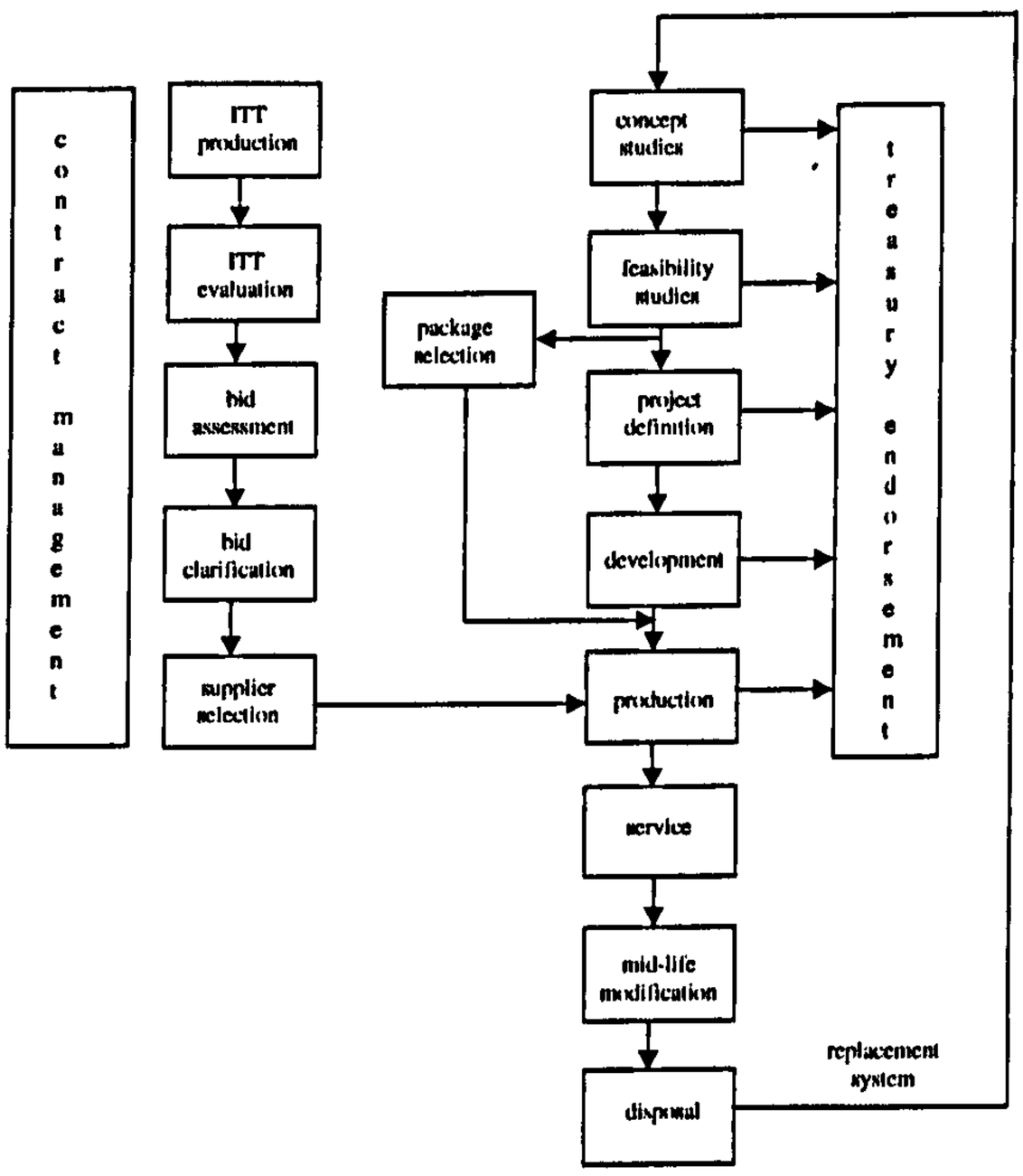


Figure 3.2: The typical high-level procurement process of Organisation A extracted from the organisation's documents and data gathered during the interviews. The process is characterised by a continuous justification to the treasury for funding at every stage. Also there is another parallel process of the support contractor who handles the ITT.

Figure 3.3 below shows organisation B's current high level procurement process. It is sequential similar to organisation A's process. The figure shows that the management and market analysis processes are performed in parallel to all other processes. The market analysis is done by a dedicated team who continuously scan the market for new products that are available and then inform the user groups whenever there is a need to develop a new system.

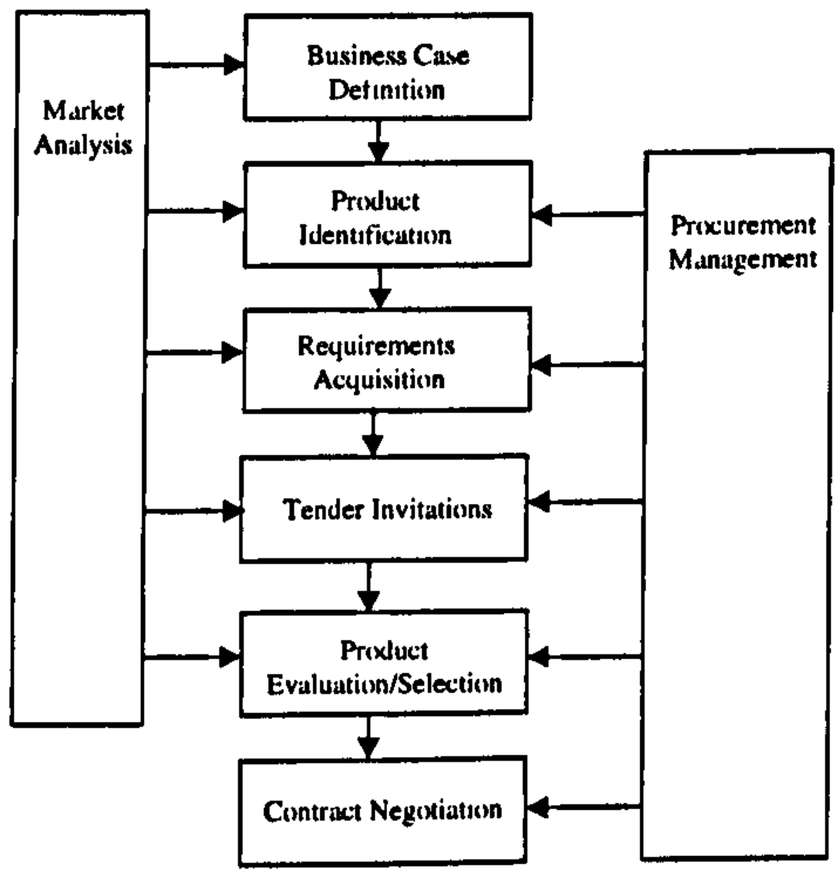


Figure 3.3: Current procurement process for Organisation B synthesised from the documents and information gathered during experts interviews.

The data gathered from the 3 organisations were structured and synthesised into a simple generic process model. The problems identified during this case study informed the design of this generic process model and its sub-processes. The following section describes the generic process model, the generic process and their sub-process and the problems associated with each generic process.

3.1.5 The generic process model

The data gathered from the 3 organisations was structured using a basic process model derived from existing literature (e.g. Finkelstein et al. 1996, Konito 1996, Tepandi 1995). The data was synthesised into a generic process model that is intended to be applicable to many software product procurement domains both in the public and private sector. The generic process model describes the most fundamental processes undertaken during product procurement. Processes and problems were

defined at 3 levels, the universal (U), worldly (W) and atomic (A) levels according to Watts' (1989) process model:

- universal (U) level processes describe general guidance for actors in the process. Each describes a uniform sequence of processes;
- worldly (W) level describes processes that are more relevant to requirements definition for systems procurement, since each guides the sequence of tasks such as procurement tasks. In their operational form, the W-level processes look like procedures which define who does what, when and where;
- atomic (A) level processes are specific to individual methods, procedures, techniques and tools which enable W-level processes.

The main purpose of the U model is to describe the basic COTS product procurement process steps and to provide general guidance on the roles and order in which the processes can be performed. The U model assumes a relatively uniform orderly sequence of steps that present a general process flow and a high level overview of understanding the procurement process. The processes of the U model are progressively decomposed into W and A levels of detail that are needed to guide the procurement teams and to represent what is really being done.

Figure 3.4 depicts the 6 generic U-level processes: *managing system procurement, requirements acquisition, supplier selection, software package selection, contract production and package acceptance*. Figure 3.4 also shows a sequence that is often adhered to, although not all the processes are performed in each procurement. For example, software package selection does not take place if a bespoke system is being procured. Management takes place throughout the procurement process. Other process can also be concurrent, for example supplier selection and software package selection processes often take place together.

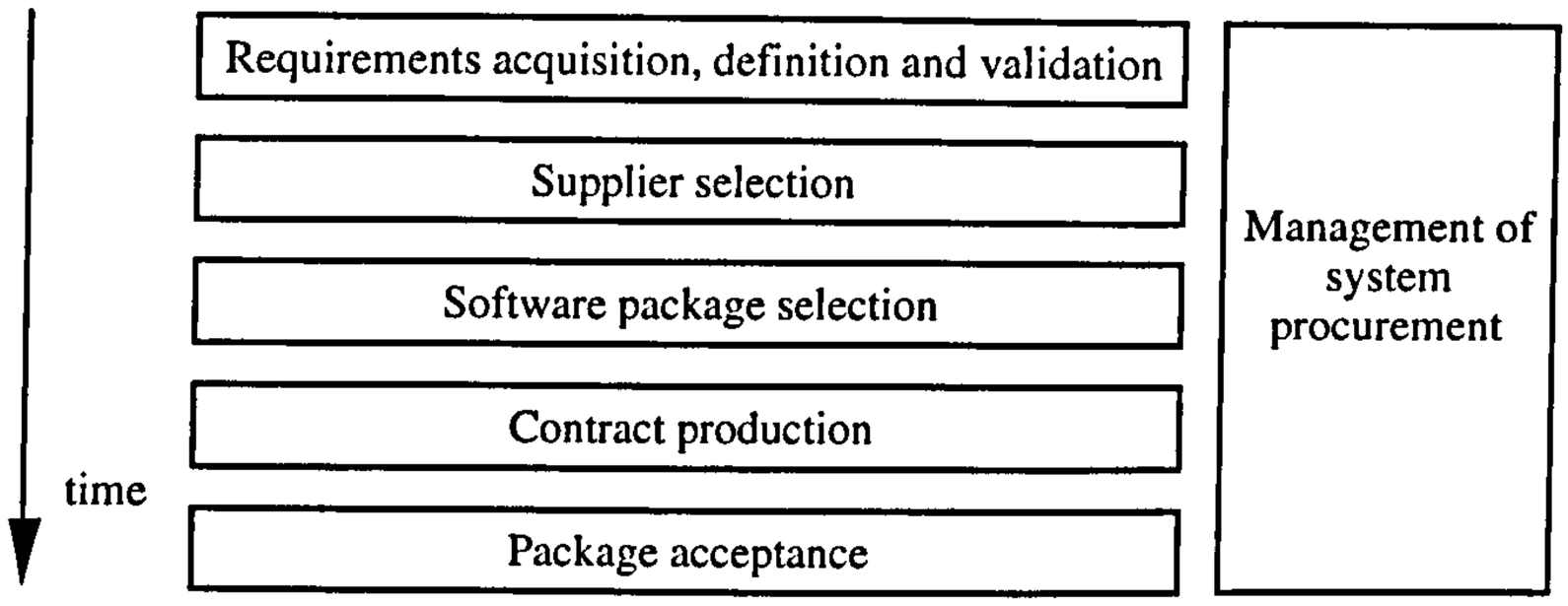


Figure 3.4: The synthesised model process used to structure the interview data

Each U level generic processes are described in the following two sections. Section 3.1.5.1 provides a brief description of each process and section 3.1.5.2 gives a detailed description of each process with its associated typical problems.

3.1.5.1 A brief description of the U-level generic processes

This section gives a brief high level description of each U level generic process.

Management of system procurement – this is the process from the concept stage to when a product or system is selected. The management team itself may be contractor personnel as in most cases with Organisation A or may be from an internal cross-functional team as in organisation B or a combination of organisation and contractor personnel as is sometimes is the case in both organisation A and B. What is important is that the management team should be set up first and very early in the process. Ideally, the team members should have between them all the necessary skills and experiences needed in the whole procurement activities and should remain with the project all the time. The team should also ideally have a stakeholder representative with them and should have access to people with specialist knowledge, skills or expertise.

Requirements acquisition – this process determines or identifies the customer's core requirements and accesses the critical system issues that the selected product should meet. Major activities are the determination of user or operational requirements, core system requirements, architecture requirements, supplier and contractual

requirements. The process is managed by the procurement management team. The requirements identification team itself may be a contractor team as with organisation A or a internal cross-functional team as with organisation B, or a combination of both.

Supplier selection – this process establishes supplier evaluation criteria, evaluates supplier proposals and ranks them for selection. The supplier evaluation team can be selected from the management team and teams can be established for each proposal area (i.e. technical, cost estimation, management skills, etc.). Ideally each evaluation team should possess all of the necessary skills and be knowledgeable about the area they are evaluating. It is important for the team to document the selection criteria and the basis and rationale for selecting the preferred supplier. This process can be performed in parallel with other processes, e.g. package selection.

Package selection – the package selection process determines whether a product will meet customer's core requirements. The selection team determines the product evaluation and selection criteria based on the customer requirements. The selection process itself will require learning enough about the candidate products in order to determine the best product that meet the customer's core requirements with minimum risks. By the end of this process, primary and back-up products would be selected. The evaluation criteria and rationale for selecting or rejecting a product should be documented.

Contract production – when a product that meets the customer's requirements has been selected or a supplier that meets the evaluation criteria has been identified, licensing and contractual agreements are then negotiated. At the end of this process a contract is produced that covers all aspects of the product procurement including terms and conditions, costs, payments, technical support, upgrades, contract termination terms, and in case the supplier goes out of business what should happen. Conditions, criteria and rationale for awarding the contract should be documented.

Product acceptance – the objective of this process is to confirm that the procured product meets end-user requirements and expectations. The product can be deployed at the customer site and end-to-end testing performed in a realistic setting. During this stage the operational readiness of the product will be reviewed. The review will focus

on functional completeness, product reliability and performance. Depending on whether the product meets the acceptance tests, it will then be accepted into service or the supplier may be requested to carry some minor modifications in order for the product to be accepted.

The results reflect the different foci of the 3 organisations. Organisation-B purchases commercial off-the-shelf systems to integrate with existing systems. Procurement is undertaken by system development staff. In contrast, organisations A and C have complex infrastructures which organise and manage the procurement of complex software systems. Different procurement tasks are undertaken by different roles. The influence of these differences on procurement processes and problems are reported later in this section. Feedback from participants at the end of interviews on the U-level process model was positive, in that all participants agreed with the definition and importance of each U-level process.

The W-level processes and problems are presented using informal descriptions rather than formal notations. The following section provides detailed descriptions of each process together with the typical problems associated with each process.

3.1.5.2 Detailed description of the W and A level generic processes

This section describes in detail the W-level processes presented in figure 3.4. Important problems that arise during each process are listed. Each problem is coded either A, B or C to indicate which organisation experienced it.

Management of system procurement - the following typical problems were found for this process:

Problem P1.1: poor relations between supplier and customer (A, C). This problem is greater in public sector procurement because supplier-customer relations are often not recognised as important. In contrast, most private sector organisations now realise the benefits of developing long-term partnerships with suppliers.

P1.2: lack of planning (A, B, C). Procurement often fails because of a lack of planning which incorporates sound procurement practices. Reasons for this include lack of relevant management experience and inadequate procedures. The result is poor procurement processes, often leading to selecting products that fail to meet customer requirements.

P1.3: failure to adhere to planned procurement processes (A, B, C). External pressures to deliver the system in a certain time period often lead to abandonment of the original procurement plan. In particular, important processes perceived as non-critical, such as supplier demonstrations, are missed in order to meet more critical deadlines. The consequence is failure to gather all relevant information, which in turn impoverishes decision-making during supplier and package selection.

P1.4: failure to obtain approval from all stakeholders (B). Important stakeholders in the required system, such as those who finance the procurement, are not always part of the procurement team. As a result it is difficult to obtain their agreement for important decisions, hence procurement is sometimes stalled or even abandoned.

P1.5: enforced justification and approval at each stage of the procurement process (A, C). Current procurements are often organised so that justification and approval has to be sought before each task. This inhibits effective planning and causes delays.

Some participants recommended other processes that might avoid the reported problems. These processes include:

- more familiarisation with the organisation's procurement plans (A, B);
- identification and documentation of potential risks during procurement (A, B);
- definition of milestones and deliverables with the supplier before procurement starts (A);
- putting change control mechanisms in place to approve and fund changes to the original procurement plan (A).

According to participants, management of the procurement process can be improved through the inclusion of people with diverse roles (B). These include people with extensive experience of the problem domain and commercial products, managers to co-ordinate procurement, and trouble-shooters who look for hardware and architecture problems. Their participation is not encouraged at the moment (B).

Requirements acquisition, definition and validation - the following typical problems were found for this process:

P2.1: failure to acquire contractual requirements (A, B, C). Contractual requirements refer to all requirements which are not about the product, for example requirements about the supplier and the legal contract with the supplier. Most requirements engineers are unaware of the importance of contractual requirements in system procurement and fail to acquire them.

P2.2: contractual requirements often conflict with other requirements on the product (A, C). This makes it difficult to produce a complete and consistent requirement specification.

P2.3: customers do not have the contractual right to change the procured software system once it has been delivered (A, B). However, requirements often change due to external factors such as new legislation. The supplier often charges the customer for such changes, thus increasing development costs.

P2.4: failure to determine core system requirements first (B). Core requirements are those requirements that are critical to success of the system and often do not change. However, organisations often fail to differentiate between core and non-core requirements.

Requirements acquisition is often similar for bespoke, procured and off-the-shelf systems. However, off-the-shelf systems provide additional opportunities. Useful processes to consider are:

- determine core system requirements first (A, B);

- determine cornerstone products (B);
- determine the current system architectures and map requirements to them (B). The architectures determine how selected packages might be integrated with existing systems;
- meet other customers and users, even those in other applications and industries who have similar requirements and technologies, as they can be useful sources of new requirements (B). Also contact vendors for demonstrations and use the internet to gather information (A, B);
- understand the system requirements in the context of the off-the-shelf systems available (A, B). Any requirements that will require custom-built software should be identified (B). This product-led requirements acquisition is often used for commercial, non-software products (B);
- acquire requirements about the supplier and the contract as well as the product (A, B, C).

During the interviews participants gave examples of important contractual requirements to acquire. Supplier requirements include information about the supplier organisation (number of employees, annual turnover, stability (A, B)), its customers (number, application domain experience (B)), training available and supplier-customer relations (previous purchases, strategic alliance, relations to individuals in the organisation (A, B)). Requirements about the contract include the degree of change possible (price, timetable, delivery (A)), the payment method (fixed, installments (A, C)) and the nature of the contract (A, C). Such information provides a basis for guiding contractual requirements acquisition (C).

Supplier selection - the following typical problems were found for this process:

P3.1: the invitation-to-tender (ITT) document given to suppliers often contains vague information (C). This is because the people who produce the ITT often do not understand requirements. This allows suppliers to interpret requirements to their own benefit and to the detriment of the customer.

P3.2: syntactic analysis of the ITT document by suppliers (C). This leads to failure by the supplier to meet all actual customer requirements. The legal keyword is "shall" because it imposes legal obligations on the supplier to meet these requirements. As a consequence suppliers often attempt to change "shall"s to "will"s. This problem is often compounded by the customer' failure to understand such legal keywords.

P3.3: there is often a failure to isolate requirements more liable to change (C). This is because changing requirements are often not identified before producing the ITT document.

P3.4: supplier selection criteria are often too cost-driven (A, B, C). This is in order to meet financial constraints. The result is that systems that meet more customer requirements are not selected.

P3.5: requirements engineers often lack expertise in bid assessment and supplier evaluation (A, C). This is due to a lack of relevant training.

P3.6: bid assessment criteria are sometimes distorted (A, B, C). Decision makers have hidden motives and agenda, and as a consequence might distort criteria, weightings and their application during supplier selection.

P3.7: supplier selection criteria are often too simplistic (A, C). This is because people lack guidance and expertise in supplier selection. For example, one criterion is whether the supplier has developed a similar system before, regardless of whether or not the system developed was successful. As a consequence the same unsuccessful suppliers are rewarded.

P3.8: there is a lack of guidelines for assessing supplier capabilities (A, B, C). For example one indicator worth considering is the supplier's ranking according to the supplier's capability maturity. However, lack of guidelines means that supplier assessment is problematic.

P3.9: supplier selection criteria are designed to maintain the status quo (A, C). Selecting suppliers regardless of previous success is an often-used criterion that maintains the status quo.

The interviewees proposed the following processes to overcome these problems:

- establish cross-functional evaluation teams for different areas of the evaluation including, for example, user representatives and external experts on candidate suppliers and products (B). The team should have the pre-requisite combination of skills, knowledge and experience to evaluate proposals (A, B, C);
- determine technical evaluation criteria (C). Use technical requirements as a baseline for determining a list of candidate suppliers with a background matching the technical requirements (A, B, C). Look at both the supplier's technical credentials, personnel, background, experience, availability and present workload (A, B);
- determine management evaluation criteria, i.e. how will the supplier manage the project (A). Look at the supplier's past performance reports, commitment to the proposed project and key personnel (A, C). Consider how the supplier will plan and control costs (A, C);
- determine cost evaluation criteria (A, B). Look at missing and uncosted elements and estimate whether costs are as expected (A). Evaluate candidate suppliers' financial credentials with regard to financial capabilities to perform the task, current financial position and credit performance (A);
- rank bids according to the above technical, management and financial requirements (A, B). Select a manageable number of bids (A). Negotiate with each supplier to obtain the most favourable deal (A). Use of sophisticated decision making techniques which recognise the complexities of the decision-making process (B);
- document supplier selection recording the basis and rationale for the selection (B). Record individual bid evaluations and debrief unsuccessful bidders (A). Inform them of selected bids and criteria for selecting them (A).

Software package selection - the following typical problems were found for this process:

P4.1: important decision makers are not involved in all activities during software package selection (B). The selection process can involve large numbers of stakeholders, all of whom should reach some level of consensus over the final selection. However, due to the failure of key decision makers to be involved throughout, consensus and agreement are difficult to achieve. Indeed, decision makers are sometimes biased towards software packages which they have reviewed over those which they did not see.

P4.2: software package selection is often too short-term and does not consider longer-term implications of the selection (B). Organisations lack long-term procurement and purchasing policies. One result is being locked into the incorrect suppliers on the basis of requirements for one product. This can have important implications for an organisation's future business processes, standards, system architectures and requirements.

P4.3: failure to consider all relevant selection criteria (A,B,C). There are few methods and techniques available to guide acquisition of all requirements needed to make informed software package selection. The consequence is that organisations purchase the wrong packages.

P4.4: customer organisations often lack experience in software package selection (B). As a result the process of package selection is impoverished.

P4.5: there are few standards for describing off-the-shelf software systems (B). As a result it is difficult and expensive to integrate off-the-shelf systems from different suppliers.

The interviewees proposed the following processes to overcome these problems:

- determine package evaluation criteria equivalent to core system requirements (B);
- use compliance with open architecture standards as an evaluation criteria (B);

- determine classes of package to be procured, to reduce the search space (B). These might be domain-independent, domain-specific or application-specific packages;
- evaluate candidate software packages against criteria equivalent to requirements (B);
- have suppliers give demonstrations of their software packages (B);
- document risks associated with each candidate package (B);
- select best-fit package(s) according to their fit with customer requirements (B).

Stakeholders who should be involved in software package selection include future end-users, managers of the procurement process and system architectures responsible for ensuring systems integration. Important criteria include the package's fit with functional and non-functional requirements, its price, its hardware configurations and the ease of integration with other systems.

Contract production - the following typical problems were found for this process:

P5.1: the supplier often aims to produce a system which meets the minimum number of customer requirements (A, C). The ITT document which provides the basis for supplier bids must be open to interpretation in order to enable a range of suppliers to bid. However this freedom gives the supplier opportunities to produce a product which meets a minimum of requirements. Furthermore, it is not in the interests of the supplier to define complete requirements. Suppliers often choose which requirements to meet.

P5.2: suppliers can profit from project over-runs (A, C). Most legal contracts include a time period during which the software package must be implemented. Conflicts over the contract enable suppliers to delay this implementation until the customer has to capitulate to obtain the purchased system.

P5.3: customers lack expertise in negotiation and contract administration (A, C). Again there is a lack of relevant procurement training for IT staff. As a result the process is not understood, not achieved with satisfaction, and ill-managed. Some organisations pass contract administration to legal departments which specialise in the

task, however this makes communication of requirements more difficult. This is because contractual experts might not understand the requirements.

P5.4: contract arrangements between suppliers and customers are adversarial (A, C). This makes systems procurement even more difficult, especially in light of increased use of fixed price and cost-plus contracts. Pitfalls are possible with both.

To overcome these problems both suppliers and customers must aim to generate a partnership that will enable shared risk-taking. Participants identified the following important sub processes:

- negotiate contract terms and conditions (A);
- negotiate conditions and criteria for awarding contract (A);
- establish upgrade and extra technical support (A, B);
- establish primary and subcontractor responsibilities (A);
- negotiate licensing agreements and legal issues (A, B);
- negotiate price and paying conditions (A, C);
- negotiate contract termination conditions (A, C).

Again stakeholders who should participate are diverse (A, B). They include procurement officials to advise on licensing and agreements with suppliers, customer and supplier purchasers who will negotiate costs, customer and supplier contract experts, legal representatives from both the customer and supplier, and those responsible for establishing contract acceptance criteria.

Package Acceptance - the following typical problems were found for this process:

P6.1: acceptance criteria are often too ill-defined to enable endorsement or rejection of the delivered system (A, B, C). Acceptance criteria must be derived from requirements to formulate questions to ask about the delivered system or scenarios for it to handle. However, there is a lack of methods and techniques for defining acceptance criteria.

P6.2: there is a lack of people with sufficient expertise in generating and checking acceptance criteria (A, C). Reasons include lack of training and failure to recognise the importance of acceptance criteria.

Acceptance checking can take a long time if modifications to the delivered system are needed. For large systems it can take several years (A). However, recommended processes to overcome this include:

- developing package acceptance criteria (A, B);
- conducting end-to-end system testing to check that the package works in an integrated system (A, B);
- conducting system operational readiness reviews (A, B);
- demonstrating the system to users (B);
- obtaining user acceptance (A, B).

3.1.6 Analysis of results

Analysis of the findings reveal several common themes in the processes and problems for procuring software systems in all the 3 organisations:

- a lack of systematic planning of product selection and requirements engineering processes, despite their complex nature (e.g. P1.2, P5.2);
- a lack of information about procurement processes, actors in these processes, possible problems, solutions to them, candidate suppliers and software packages, and few relevant useful standards (e.g. P1.3, P3.9);
- actors lack the skills and expertise for procuring software systems (e.g. P3.6, P4.4);
- a failure of involvement of important decision-makers throughout the process (e.g. P1.4, P4.1);
- a failure of actors to recognise the importance of non-product contractual requirements (e.g. P2.1);
- a failure of actors to recognise the importance of unchanging core requirements (e.g. P2.4);
- the conflict between supplier and customer (e.g. P3.2, P5.5).

It is interesting to note that, at a broad level, findings were different for each organisation. Organisation B has few guidelines for procuring its numerous COTS systems. In contrast organisations A and C have extensive experience of system procurement and complex structures to manage this procurement. This is reflected in the data and in the problems as shown in Table 3.1. Organisation-B's problems are different to those of organisations A and C. Organisation B reported fewer problems during procurement management, supplier selection, contract production and product acceptance, and more problems during package selection. This is indicative of the diverse nature of the requirements engineering problems that are encountered in a packaged-based development but that are not usually experienced in traditional development processes.

	Organisation A	Organisation B	Organisation C
P1.1:poor relations between supplier and customer	√		√
P1.2: lack of planning	√	√	√
P1.3: failure to endure to planned procurement processes	√	√	√
P1.4: failure to obtain approval from all stakeholders		√	
P1.5: enforced justification and approval at each stage of the procurement process	√		√
P2.1: failure to acquire contractual requirements	√	√	√
P2.2: contractual requirements often conflict with other requirements	√		√
P2.3: customers do not have contractual rights to change the procured software once it has been delivered	√	√	
P2.4: failure to determine core system requirements first		√	
P3.1: the invitation-to-tender document given to suppliers often contains vague information			√
P3.2: systematic analysis of the ITT document by suppliers			√
P3.3: there is often a failure to isolate requirements that are more liable to change			√
P3.4: supplier selection criteria is too often cost-driven	√	√	√
P3.5: requirements engineers often lack experience in bid assessment and supplier evaluation	√		√
P3.6: bid assessment criteria is sometimes distorted	√	√	√
P3.7: supplier selection criteria are	√		√

often too simplistic			
P3.8: there is lack of guidelines for assessing supplier capabilities	√	√	√
P3.9: supplier selection criteria are designed to maintain the status quo	√		√
P4.1: important decision makers are not involved in all activities during software package selection		√	
P4.2: software package selection is often too short-term and does not consider long-term implications of the selection		√	
P4.3: failure to consider relevant selection criteria	√	√	√
P4.4: customer organisations often lack experience in software package selection	√	√	√
P4.5: there are few standards for describing off-the-shelf software systems	√	√	√
P5.1: the supplier often aims to produce a system which meets the minimum number of customer requirements	√		√
P5.2: suppliers can often profit from project over-runs	√		√
P5.3: customers lack expertise in negotiation and contract administration	√		√
P5.4: contract arrangements between suppliers and customers are often adversarial	√		√
P6.1: acceptance criteria are often too ill-defined to enable endorsement or rejection of the delivered system	√	√	√
P6.2: there is a lack of people with sufficient expertise in generating and checking acceptance criteria	√		√
	22	15	24

Table 3.1 Overview of the distribution of the problems experienced by organisations A, B and C. The table shows that organisations A and C experienced almost 90% of the problems.

3.1.7 Validation of the findings

Similar problems were found in the failed implementation of the London Ambulance Service's (LAS) Computer Aided Dispatch (CAD) system (Dowell & Finkelstein 1996). The following section uses this case study to demonstrate the problems identified from organisations -A, -B and -C.

The LAS CAD system was developed to replace manual procedures to provide command and control functions and management information. It consisted of a combination of CAD software and hardware, an electronic gazetteer and mapping software, communications interface software, radio systems, mobile data terminals and an automatic vehicle location system. Its success was dependent on accurate information, reliable technologies and full co-operation of all users. It aimed to automate as much of ambulance command and control as possible. However, the system collapsed and was abandoned in November 1992.

Reasons for system failure included poor requirements definition and procurement. When the LAS management distributed an ITT document, 35 suppliers responded expressing interest and received a full requirements specification from the LAS. In response 17 suppliers submitted firm bid proposals. LAS management compared these proposals and one consortium (Systems Option, Apricot and Datatrak) was selected. However failures in the tender and bid evaluation process contributed to the overall system collapse. The procurement process was ill-planned and ill-managed (P1.2) and subject to outside financial and political pressures to improve LAS performance (P1.3). Changes in the procurement team made it difficult to achieve consensus decision-making (P1.4). Furthermore the team had inappropriate skills for the task (P1.5). During requirements acquisition there was a failure to acquire contractual requirements (P2.1). Indeed the specification was too prescriptive and left no scope for flexibility in system design. There were no standards available to structure the requirement specification for procurement purposes (P4.5).

During package selection, criteria for both supplier and software package selection were too simple (P3.4). Criteria from the ITT document were vague (P1.4). Cost was the overriding selection criterion (P3.4). The system manager and contract analyst responsible for software package selection lacked relevant experience (P3.6). Claims made from the suppliers about previous successful system developments were misleading (P3.9). The lack of expertise in contract production led to poor contract development (P5.3, P5.4). As a result contractor responsibilities were not established. Contract termination conditions were not negotiated and there was no established contract acceptance criteria (P1.6). When the system was delivered there was little evidence of system acceptance testing (P6.1, 6.2).

In short, the LAS CAD fiasco demonstrates the consequences of poor requirements engineering for COTS product procurement. It supports findings reported above and indicates that the problems are widespread in practice. The LAS fiasco also indicates that the problems exist at many organisational levels and each problem is associated with a specific organisational level.

The problems reported above are not addressed by currently existing COTS development methods. Also most current requirements engineering do not explicitly address most of these problems as shown in table 3.2.

	OTSO	CISD	IIDA	IusWare	Feature Analysis	Volere	Rational Process	SSADM	SSA
P1.1									
P1.2	√			√			√		
P1.3								√	
P1.4									
P1.5									
P2.1									
P2.2									
P2.3									
P2.4	√		√		√	√	√	√	√
P3.1									
P3.2									
P3.3						√	√	√	
P3.4	√								
P3.5									
P3.6									
P3.7	√				√				
P3.8									
P3.9									
P4.1				√					
P4.2									
P4.3	√			√	√				
P4.4									
P4.5		√							
P5.1									
P5.2									
P5.3									
P5.4									
P6.1									
P6.2									

Table 3.2. Most problems identified in table 3.1 are not explicitly addressed by current COTS software development and requirements engineering methods. New methods that address these problems are therefore required.

As shown in table 3.2, developing systems from COTS software presents customers with new problems that that are not normally experienced in traditional development processes (H1). The study reported above shows that there is need for new methods to address the new problems and therefore hypothesis 1 (H1) is accepted. However, to further test hypothesis 1, a second study was undertaken to acquire requirements for selecting a COTS product and to evaluate and select the preferred product. The lessons learned during this study, the problems experienced and the proposed solutions to the problems informed further design of the new method. The study also identified techniques that are needed for this process. The study is described in the next section.

3.2 Acquiring requirements for COTS product selection

3.2.1 Study method

The customer was the MoD Procurement Executive (PE). The PE wanted new methods and software tools with which to manage requirements for a new naval platform which would take 20 years to develop. The aim of the study was to acquire requirements about and recommend commercial requirements engineering methods and software tools for trial by the PE.

The study had 11 weeks to make a recommendation to the PE. It was decided that, where possible, commercial product selection procedures would be adhered to. No research ideas or techniques from disciplines outside the fields of systems or requirements engineering were used.

Requirements acquisition took place from both MoD documents and stakeholders. Five meetings with between 6 and 10 stakeholders took place over a 3-week period. Each meeting was divided into a review of the current requirements document and acquisition of new requirements from the stakeholders. The final requirements document contained 133 atomic requirement statements.

To save time, market research to identify candidate products was undertaken in parallel with requirements acquisition. An initial version of the requirements document was transformed into a questionnaire and sent to over 30 candidate

suppliers to determine the coverage of their products, (see Appendix 3a). From the supplier responses, a shortlist of 6 candidate products was produced. Next, a set of 35 complex test cases for product evaluation was developed from the final requirements document, (see Appendix 3b). A prototype was developed to test the accuracy and viability of the test cases. All the shortlisted suppliers were invited to demonstrate their product against the 35 test cases, but only 5 attended the demonstration sessions. During each evaluation session, 3 members of the team using both quantitative scores and qualitative comments recorded product feature compliance with each requirement. After each evaluation, the team members agreed the final product-requirement compliance scores. These scores were then investigated using methods such as weighted scores to produce relative ranking of the products (see Appendix 3c). As a result, trial use of two requirements engineering software tools was recommended.

3.2.2 Results

The requirements engineering team experienced many problems. In particular, the requirements acquisition and product selection processes were problematic, even though the customer was content with the recommendations. As hypothesis one (H1) states, the lessons learned and problems encountered are seldom experienced in the traditional development processes. The next section outlines the lessons that were learned and 11 of the many problems that were encountered and proposes possible solutions to each of the 11 problems. The problems that were encountered and proposed solutions to these problems inform the design of the new method.

3.2.2.1 Lessons learned, problems encountered and suggested solutions

This study provided a range of lessons and problems about the nature of acquiring requirements for COTS software product selection.

Lesson-1: acquire in more detail those requirements that enable effective discrimination between products.

Problem: too much time was spent acquiring and modeling requirements that were met by all 5 evaluated products. Most of these requirements did not enable effective discrimination between the products. In contrast, the requirements that enabled

effective product selection were not modeled in sufficient detail. For example, requirements about the product's compatibility with Microsoft Word™ and Access™ products were less detailed but were more important and key to the selection of a product.

Solution: requirements acquisition and product evaluation should be both iterative and concurrent. The selection team can then become familiar with requirements and products at the same time, thus making both requirements acquisition and market research more flexible and responsive. This can be achieved using a range of techniques. One such technique is card sorts. Card sorts are simple to use and can be used to acquire requirements which discriminate between products (Maiden & Rugg 1996). The requirements engineer writes candidate product names on 3"x5" cards and asks stakeholders to use the cards to sort the products into categories. Criteria for these sorts (e.g. "compatible with Microsoft™ Word") indicate customer requirements which discriminate between products. Product categories (e.g. "compatible" and "not compatible") indicate product compliance to these requirements. A useful variation, card sort triage, requires the stakeholder to describe the similarities between two products and their common differences with a third product. Card sort techniques also fit well with laddering (Rugg & McGeorge 1995), in which the stakeholder is asked to describe common categories and classes of products and their features to discover important but non-discriminating customer requirements, thus avoiding time-consuming acquisition of less important requirements. Discriminating requirements then provide a starting point for more thorough requirements acquisition using other techniques such as those reported in Maiden & Rugg (1996).

Lesson-2: requirements must be as measurable as possible to enable effective product selection.

Problem: for most requirements it was difficult to measure product-requirement compliance, thus making product selection problematic. One reason was that requirements were not verifiable, in that their fit criteria were not expressed as logical expressions or quantifiable tests recommended by current commercial requirements standards (e.g. Mazza et al. 1994).

Solution-1: making requirements measurable is very difficult than its usually perceived. The recommended solution is to pre-empt how the requirements will be used during product selection, for example as questions in questionnaires sent to

suppliers, or as evaluation test cases during on-line product demonstrations, then tailor the verifiable fit criteria for these requirements accordingly. However, this cannot always be done effectively for a large number of customer requirements, so the focus should be on requirements which enable effective product discrimination. An iterative approach is essential for success: use techniques reported in lesson-1 to determine requirements which discriminate between products, define fit criteria for these requirements, then re-evaluate product-requirement compliance using these criteria. Several iterations might be needed to determine precise and measurable fit criteria for the requirement, to evaluate the degree of product compliance to the requirement.

Solution-2: another solution is to use contrived acquisition techniques to acquire quantitative scores for product-requirement compliance. One such technique is repertory grid analysis, in which stakeholders are asked for attributes applicable to a set of entities and values for cells in an entity-attribute matrix. Advantages from using this technique include the representation of requirements in a standardised, quantifiable format which is even amenable to statistical analyses as a basis for justifying product selection decisions.

Lesson-3: use software prototypes to aid generation of test cases for product evaluation.

Problem: generation of measurable test cases is difficult without prior knowledge about candidate products or prior extensive experience of test case generation.

Solution: the solution is to use a software prototype to aid generation of test cases. If a software prototype exists, the selection team can undertake mock evaluations of the prototype using first-draft test cases as if the prototype was a candidate product. This may lead to some refinement of the test cases themselves, so as to make them more measurable. This can enable the evaluation team to determine the correct responses to complex data base queries used during product evaluation. However, most projects will have to develop such a prototype. Rather than generate a single, integrated prototype, a more cost-effective approach might be to generate several smaller, partial prototypes or concept demonstrators of desirable but discriminating product features with which to develop verifiable fit criteria.

Lesson-4: structure the requirements in way that makes it easy to formulate test cases.
Problem: the hierarchical structure of the requirements is incompatible with the sequential structure of the test cases, and this makes test case generation difficult. For example, the requirement that the product be configurable to customer needs was included in all 35 test cases in order to evaluate how configurable all of the functions of each product were. However, this weakened the link between requirements and test cases, and made test case management more difficult.
Solution: the obvious solution is to acquire requirements using use cases and scenarios so that the requirements are more amenable to test case generation. For example, Graham's (1996) SOMATiK approach proposes seamless decomposition of goals into tasks which achieve these goals, then generation of use cases which are equivalence classes of task scripts and of scenarios which are equivalence classes of a use case (p131). SOMATiK's seamless transformation is both practical and effective. Graham reports its successful use on over 20 projects at Swiss Bank Corporation. The SOMATiK software tool even generates simple software prototypes which can be useful during test case generation, thus also providing at least a partial solution to problems reported in lesson-3.

Lesson-5: the scope of the product under evaluation is difficult to define.
Problem: requirements management tools are complex and depend on other products such as data base management systems. In order to undertake a complete product evaluation, these other dependent products have to be evaluated which in turn makes the evaluation more complex and time-consuming.
Solution: LORAL's revision of the spiral process for COTS product selection (Walters 1995) recognises the need to select cornerstone products first, then integrate other products around them. However, guidance for detecting and selecting these products is still needed. One simple solution is to use checklists to see whether customer requirements are essential, stable and urgent, and whether each candidate product will have a long period of use, meets essential rather than non-essential requirements, needs modification, is currently available, and adheres to current product standards. It is also important to determine dependencies between products, for example "Can this function be achieved without additional products?". If the answer is "no", ask follow-on questions such as "Are the additional products available with the core product?", "Where are the additional products available from?" and

"How much configuration of these products is needed?". Such questions are embedded in new method's templates to encourage the requirements engineer to ask the right question at the right time during requirements analysis.

Lesson-6: stakeholder representatives should be present during product evaluation.
Problem: sometimes there is a need to ask detailed questions during the evaluation session using information about the problem domain. Such problem domain information may not always be acquired from stakeholders prior to evaluation or recalled by the team members during an evaluation session.
Solution: the solution was to have a stakeholder representative with a good understanding of the problem domain available during each evaluation. This gave the team the advantage of being able to ask more detailed questions during each evaluation.

Lesson-7: techniques are needed to record information during product evaluation.
Problem: the large number of requirements made product-requirement compliance a complex task. The team made over 1500 compliance decisions during a total of 18 hours of product evaluation. However, it did not use techniques to record the rationale for these decisions. This made agreement of product-requirement compliance scores within the team members after each evaluation session very difficult because all the reasons for all of the scores have not been recorded.
Solution-1: record rationale for product-requirement compliance scores during the evaluation using, for example, design rationale techniques such as discussed in Moran & Carroll (1996) and demonstrated in Figure 3.5. To save time, each product-requirement compliance test, the properties of each product and the requirement statements can be entered into the tool before evaluation begins. The use of a scribe who is independent of the evaluation process is recommended to record rationale during each evaluation. Video tapes of the evaluation session are another means of recording information. Video tapes can even be linked to design rationale diagrams by including a time reference to when a product-requirement evaluation took place.
Solution-2: detecting dependencies between product features is essential for effective product selection, so rationale diagrams can be extended to show key dependencies which inform selection. Figure 3.5 shows that two product features (access rights for individual users, tailorable user profiles) belong to two versions of the same product.

Since the team has to choose one version or the other, a logical operator 'OR' is added to the diagram. Such dependencies have an important impact on the selection of decision-making techniques, as reported for lesson-8.

Solution-3: another solution is to use feature analysis techniques from non-software product procurement (e.g. Kitchenham & Jones 1997) to obtain product-requirement compliance scores in a more systematic way. The DESMET approach (Kitchenham 1996) proposes taxonomies of desirable features derived from expertise of DESMET consortium members to draw on during product selection. The DESMET approach treats each product feature as independent of any other feature of the product, however this is not often the case for even simple software products, so this approach should be used with care.

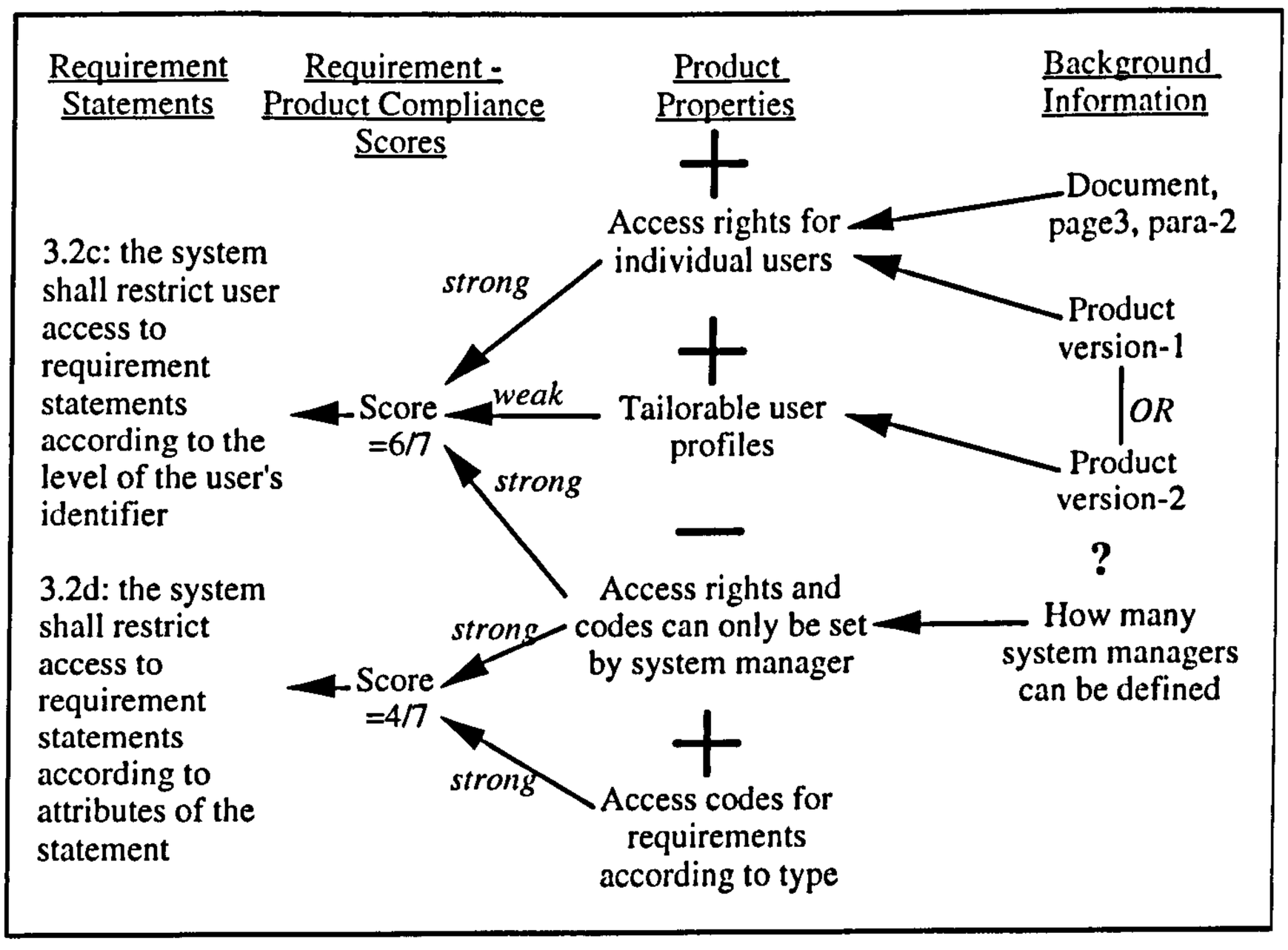


Figure 3.5: Example of a design rationale tailored for product evaluation. It enables the user to record product-requirement compliance scores and their reasons, as well as references to other scores, requirements and product properties. For example, the product was awarded a score of 6 out of 7 for compliance to requirement 3.2c. Positive reasons for this score include the product's provision of access rights for individual users and tailorable user profiles. However these access rights and codes can only be set by the system manager. Furthermore, different product properties are available from two

different versions of the product. The 'OR' link between them indicates that only one of the two versions can be selected. This, in turn, influences both the selection to be made and the techniques to use to aid the making of that decision.

Lesson-8: weighting requirements for product selection can be problematic.

Problem: one stakeholder weighted all of the customer requirements using simple percentage scores. However, these weightings were sometimes inconsistent and led to confusion about what were the most essential customer requirements.

Solution: use more sophisticated requirements weighting methods such as multi-criteria decision-making techniques. One such technique, the Analytic Hierarchy Process (AHP) has received some interest in the requirements engineering (Karlsson & Ryan 1997) and software engineering (Kontio 1996) communities. AHP was developed for multiple criteria decision making situations. It supports hierarchical structures common when modelling system requirements. Rankings and weightings are obtained through paired comparisons of requirement statements which are converted to normalised rankings using the eigenvalue method, which means that the relative rankings of alternatives are presented in ratio scale values which total one (Saaty 1990). However, one strong assumption for use of the AHP is that all criteria (i.e. requirements) are independent. This was not the case in this case study. Indeed, it is not the case for most system requirements, therefore results from AHP analyses might be unreliable when there are dependencies between system requirements. One alternative solution is to use the outranking methods. Outranking methods seek to enrich dominance relations between criteria without having to make the strong assumptions needed for the AHP (Fenton 1994). There are several formal definitions of outranking methods, however all involve building the outranking relation then exploiting it with regard to the current problem.

Lesson-9: weighting requirements for product selection can be time-consuming.

Problem: thorough use of AHP as recommended in Saaty (1990) would have required, on estimate, over 42000 individual paired comparison scores. Clearly, time constraints on product selection would have made this impossible.

Solution: use the AHP for specific purposes only. To avoid a combinatorial explosion in the number of individual paired compliance scores to be made, use multi-criteria decision-making techniques to weight customer requirements but not to determine

product compliance to these requirements. When doing this, only use the AHP if underlying assumptions for its use are met, that is there are few if any interdependencies between customer requirements (i.e. the criteria) or between product features (the alternatives). Furthermore, to ensure its cost-effective use, use the AHP when other, simpler decision-making techniques are inappropriate or when a more sensitive analysis is needed to resolve disagreements or to support critical decisions about discriminating product features (lesson-1) or cornerstone products (lesson-5). Figure 3.6 shows use of the AHP technique to weight top-level requirements. However, a word of caution is needed: AHP should be used for weighting requirements but not product-requirement compliance. This is due to very large number of paired comparisons needed for product selection. During the study it estimated that over 42000 individual paired comparison scores would have been needed to use the AHP technique for product selection! Clearly time constraints do not permit this.

top-level requirements

	Requirements Capture	Requirements Modelling	Requirements Management	Procurement Management	Technical Features	Total Scores	Priority Vector
Requirements Capture	1	0.5	0.25	1	1	3.75	0.115
Requirements Modelling	2	1	0.5	3	1	7.5	0.229
Requirements Management	4	2	1	3	2	12	0.366
Procurement Management	1	0.333	0.333	1	0.333	2.999	0.092
Technical Features	1	1	0.5	3	1	6.5	0.198
					Totals	32.749	1

the number 4 indicates that requirements management is four times more important than requirements acquisition

final requirement weightings

Figure 3.6: A sample of using the AHP method to weight requirements. However, care must be taken when applying the techniques, as there is a possibility of having a large number of pair-wise comparisons.

Lesson-10: product evaluation is a team game, so it should be treated as such.

Problem: biases are always possible when scoring product-requirement compliance. One post-evaluation analysis of individual and agreed product-requirement compliance scores revealed a trend towards agreement with one team member more than the others, possible due in part to the different experiences of the members.

Solution: one solution is to use reference products with which all participants are familiar, such as commercial products or, in the team's case, the in-house requirements engineering tool. Such products enable the evaluation team to calibrate product-requirement compliance scores before undertaking each individual evaluation test case. However, no one reference product will always have all of the product features to be evaluated, so be prepared to use several reference products to calibrate scores, although this does mean that the evaluation will often take more time. Card sort triage techniques from lesson-1 can also be adapted to ask for quantitative measures of similarity and difference between two candidate products and the reference product. It is also possible to include such reference products in design rationale diagrams (lesson-7) to record rationale for product-requirement compliance scores.

Lesson-11: beware of the supplier's sales pitch and focus on the product.

Problem: a requirements management tool is a complex COTS product. A proper evaluation needs effective demonstration by supplier representatives. However, the quality of the demonstrations varied considerably across products. However, the law of the marketplace does suggest that the supplier gets what it deserves, but this does not ensure that the customer purchases the product most compliant with its needs.

Solution: the solution was to stick to the script imposed by the test cases to ask the same questions to all candidate suppliers. More flexible follow-up questions were useful coaxing additional information out of these representatives.

3.2.2.2 Discussion

The problems reported above are not experienced in the traditional development process. These problems are also not addressed by currently existing COTS-based development and requirements engineering methods. Therefore hypothesis 1 (H1) is supported.

However, in spite of the reported problems, product selection was successful and the customer was content with the recommendations. However the product selection process could have been improved. The reported problems as well as those not reported here are used to propose the simple techniques to improve requirements acquisition for COTS selection. The lessons learned provide new and important knowledge about selecting COTS software products. In turn this knowledge and the suggested solutions to the problems are brought together in the design of the first version of a new integrated, template-based COTS development method that is aimed at addressing the problems. This method is described next.

3.3 PORE: A requirements acquisition method for COTS-based systems development

To investigate hypothesis 2:

H2 It is possible to design more effective methods which directly address current problems in requirements engineering for COTS-based development, a new method called PORE is developed. The PORE (Procurement-Oriented Requirements Engineering) method integrates techniques for requirements acquisition and product selection with process guidance for choosing and using each technique. The method draws on techniques from different disciplines already indicated in some of the 11 lessons learned and reported in section 3.5.2:

- knowledge engineering techniques such as card sorting and laddering (e.g. Rugg & McGeorge 1995) which are useful when acquiring information about categories of products, suppliers, procurement contracts and hierarchical information about product properties as well as the requirements themselves, see lessons 1 and 2;
- techniques from feature analysis (Kitchenham & Jones 1997) to aid when scoring the compliance of each product to each requirement, see lesson 7;
- MCDM techniques (e.g. Saaty 1990) and outranking methods to aid decision-making during the complex product ranking and selection process, see lesson 8;
- design rationale techniques (e.g. Buckingham-Shum & Hammond 1994) to record and aid this decision-making process, see lesson 7.

PORE also includes guidelines for designing product evaluation test cases, see lessons 3, 4 and 5, and organising effective evaluation sessions, see lessons 6 and 11. These techniques and guidelines are presented as a series of templates for requirements acquisition and product selection at different stages in the product selection process. The PORE method exists at two levels. Level 1 is the simple template level and Level 2 is the more complex process level. Section 3.3.1 below discusses Level 1. Level 2 is discussed in chapter 4 to further investigate hypothesis 2. Figure 3.7 depicts an outline of the PORE method.

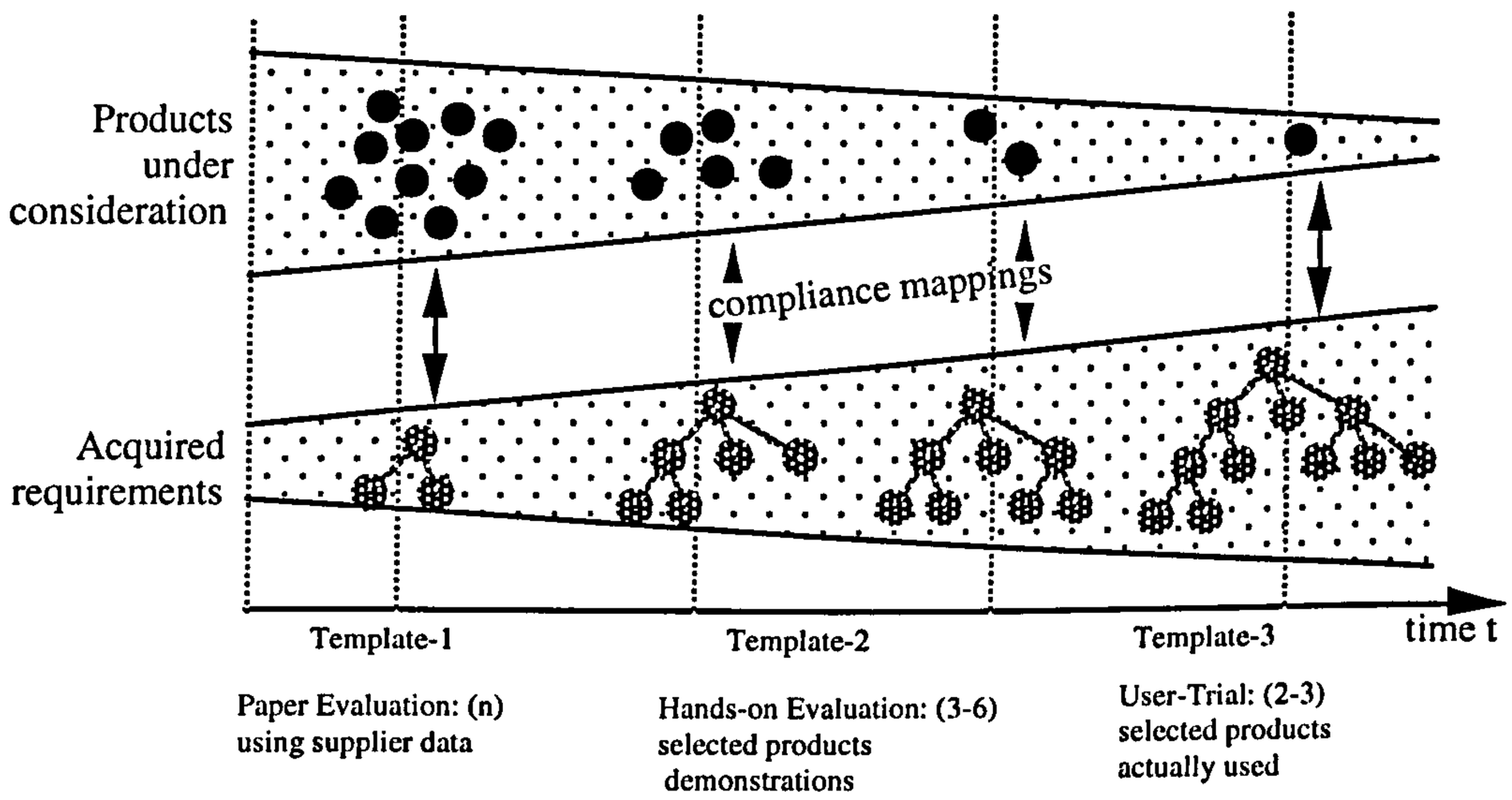


Figure 3.7: An outline of the PORE process model for product selection. At the beginning of the process there is a large number of candidate products under consideration and few customer requirements acquired. Using supplier information obtained using template 1, products are evaluated against customer requirements and those that do not sufficiently meet the requirements are rejected. As a result, the number of candidate products is iteratively reduced and the number and detail of customer requirements increases.

3.3.1 PORE templates

PORE supports iterative requirements acquisition and product selection/rejection until one or more products are compliant with a sufficient number of customer requirements. It divides this process into stages and, in the first version, provides three templates for three key stages of the process, see Figure 3.7. Each template defines the product information and customer requirements to acquire, and the techniques for acquiring this information and making decisions about it. At the beginning of the process there are few customer requirements but a large number of candidate products. Guided by the templates, non-compliant products are filtered out using different techniques that are provided in the PORE method box. Over some time, and after a number of iterations, the number of customer requirements increases and the number of candidate products decreases as products are rejected. The 3 templates are:

- Template-1, to guide the requirements engineer when acquiring essential customer requirements and product information sufficient to select and reject products as a result of supplier-given information;
- Template-2, to guide the requirements engineer when acquiring customer requirements and product information sufficient to select and reject products from supplier-led demonstrations using test-cases for individual requirements;
- Template-3, to guide the requirements engineer to acquire customer requirements and product information sufficient to select and reject products as a result of customer-led product exploration, (i.e. user trial). The full details of all PORE templates are provided in appendices 3d – 3f.

The iterative nature of the requirements acquisition and product selection processes means that each template might be used several times during a product selection process. The following three sections describe the three PORE templates in more detail. The templates are further used as basis for evaluating the PORE method in chapter 6.

3.3.2 Template-1: paper evaluation template using supplier response data

This template is to be used during the early stages of requirements acquisition and product selection, when the evaluation team relies on supplier data in sales brochures, technical documents, telephone conversations, responses to questionnaires and information on the internet, as well as internal or public market analyses. The main objectives of template 1 are:

- to provide the evaluation team with guidance when acquiring core essential customer requirements;
- to provide the evaluation team with guidance when identifying candidate products that currently exist in the market ;
- to provide the evaluation team with guidance when acquiring product and supplier information that is necessary to select and reject non-compliant products;
- to provide guidance when comparing product information provided by suppliers against the most core critical high level customer requirements for product-requirement compliance checking;
- to provide technique guidance for gathering customer requirements and product information;
- to provide guidance to initially screen many products and to shortlist one or more products for detailed evaluation.

Template 1 provides guidance for selecting techniques and ways for gathering customer requirements, product information, type of information and other information necessary for effective product evaluation and selection. It also provides guidelines and instructions to the requirements engineer or evaluation team on how to apply the template. Due to a lack of detailed and accurate data, the requirements engineers should be prepared to sometimes backtrack on selection decisions made earlier if important new information becomes available. Template 1 recommends the use of simple criteria for product selection which require quantitative information about products (e.g. how many users can use the product at the same time?) and Boolean responses to product-requirement compliance questions (e.g. is the product compatible with MicroSoft Word?). A segment of the template is given in Figure 3.8.

It outlines the types of product information and customer requirements to acquire, and simple techniques to use. The full template for this stage includes more techniques to use, guidelines for technique use and simple frames for describing product information and customer requirements of different types. The full template is described in appendix 3d.

INFORMATION AND REQUIREMENTS TO ACQUIRE:

• BASIC PRODUCT AND SUPPLIER INFORMATION;
• TECHNICAL PRODUCT FEATURES (e.g. current version number and period since last release);
• TECHNICAL SUPPORT ARRANGEMENTS for the product (e.g. licensing arrangements and cost, cost for technical support, training cost and availability, run-time fees, is source code available, extra vendor support provided and policies on upgrades and fixes);
• HISTORICAL INFORMATION about the product and supplier (e.g. how long the supplier has been in business, how long has the product been available, supplier annual turnover and customer base, number of products sold, number of employees, supplier references, track record in the business sector);
• ESSENTIAL FUNCTIONAL USER REQUIREMENTS, acquired from stakeholders and derived from candidate products.

TECHNIQUE SEQUENCE TO USE:

1. GATHER PRODUCT INFORMATION: read product documentation to gather basic product information;
2. ACQUIRE CUSTOMER REQUIREMENTS: acquire first-pass essential customer requirements using simple techniques such as brainstorming and interviewing (Maiden & Rugg 1996), if possible without reference to candidate products. Acquire functional rather than non-functional requirements, since products are easier to evaluate for functional requirements at this stage;
3. DEVELOP A QUESTIONNAIRE to ask each supplier how much this product is compliant with each of these essential user requirements. Also use this questionnaire to provide basic supplier and product information. Design the questionnaire to elicit sufficient information without it becoming too long and difficult to complete. Distribute it to all suppliers, set a deadline for replies and receive responses. Responses from suppliers should be quantitative, enumerative or boolean so that analysis of responses is simpler;
4. GET TO KNOW THE CANDIDATE PRODUCTS: familiarise yourselves with products using demonstration copies;
5. EVALUATE QUESTIONNAIRE RESPONSES to reject products which are non-compliant with essential customer requirements;
6. DISCOVER MORE CUSTOMER REQUIREMENTS from product information elicited from questionnaire responses. Often candidate products have desirable properties not discovered during requirements acquisition. These requirements should be explored with the customer at this stage. Use techniques such as structured interviews, prototype walkthroughs using product demonstration copies and scenarios of the use of the discovered requirement;
7. ACQUIRE MORE CUSTOMER REQUIREMENTS which enable discrimination between products (lesson 2). The template proposes iterative use of: (i) card sorts to acquire requirements which enable discrimination between products: (ii) rejection of products due to poor product-requirement compliance;
8. ACQUIRE DETAILED CUSTOMER REQUIREMENTS using scenarios as a basis for designing test cases for product evaluation using Template-2 (lesson 4).

DECISION-MAKING TECHNIQUES TO USE:

Decision-making at this stage is straightforward. However record rationale for product acceptance and rejection using simple decision tables and, when more complex, design rationale techniques.

Figure 3.8: Part of the template for product selection using supplier data.

3.3.3 Template-2: hands-on evaluation template

It is common to have supplier-led product demonstrations during product selection. Such demonstrations are often the first chance for the evaluation team to undertake more complex product-requirement compliance tests. Template 2 is used in the second stage of the evaluation process when conducting detailed product evaluations. Its main objectives are:

- to provide process guidance when acquiring customer requirements and product information sufficient to select and reject products from supplier-led demonstrations using test-cases for individual atomic requirements;
- to provide guidance when organising evaluation sessions of selected products when the suppliers are brought in-house for demonstrations;
- to provide guidance when designing test-cases for individual requirements that are used during each product demonstration session;
- to provide technique guidance for decision-making for recommending one or more products to the customer.

Among other things, this template guides the evaluation team in determining product's technical and functional capabilities that meet the customer requirements. It encourages the requirements engineer to explore product compliance with individual requirements. Effective preparation of the product evaluation tests is critical. One of the difficulties of product selection or evaluation is the formulation of test cases. This problem is caused by the traditional hierarchical structure of requirements which result from the acquisition phase. Before test cases can be designed, requirements need to be structured in other ways. One way is to structure the requirements by their type. One advantage of structuring requirements according to their type is that certain requirements types are mapped to certain types of product features. Therefore, the evaluation team can use requirements types to organise test cases design according to these types. The template provides process guidance on what the evaluation team should do before, during and after each product demonstration session. It places more

emphasis on technique use than on the information to acquire. Part of the template is shown in Figure 3.9. The full template for this stage is much more complex as described in Appendix 3e.

TO DO BEFORE THE DEMONSTRATION SESSION:

1. DEVELOP SIMPLE WORKING PROTOTYPES OF THE REQUIRED SYSTEM to discover and acquire further customer requirements prior to product evaluation. Use the prototype to improve and design of test cases for product evaluation (lesson 3);
2. HAVE STAKEHOLDER REPRESENTATIVES PRESENT during each demonstration to suggest previously-unforeseen requirements or to provide important domain information (lesson 6);
3. WORK WITH STAKEHOLDERS TO WEIGHT CUSTOMER REQUIREMENTS. If requirements are hierarchical use the AHP (lessons 8 & 9) on small, self-contained clusters of requirements with few dependencies to other requirements. This will avoid an exponential increase in the number of weighting decisions to be made and ensure the suitability of the AHP;
4. MAKE COMMERCIAL SOFTWARE TOOLS AVAILABLE, for example Saaty's Expert Choice and Karlsson & Ryan's (1997) tool, to calculate requirement weightings with the AHP (lesson 8);
5. PROVIDE EFFECTIVE UNITS OF MEASURE for product-requirement compliance scores through iterative refinement and evaluation of verifiable fit criteria for requirements which discriminate between products (lesson 2).

DURING EACH DEMONSTRATION SESSION:

6. ASK QUESTIONS ABOUT THE PRODUCT to determine cornerstone products first (lesson 5);
7. ONLY ALLOCATE COMPLIANCE SCORES IF THE PRODUCT PROPERTIES ARE DEMONSTRATED. Do not score unsubstantiated claims about the product (lesson 11);
8. If it is difficult to score for compliance USE REFERENCE MODELS (lesson 11). A reference model describes well-known, prototypical properties of a product and exemplar compliance scores for common, everyday tasks;
9. RECORD DECISIONS BEHIND COMPLIANCE SCORES using video and commercial design rationale software tools (lesson 7). Have an independent scribe record these rationale during the evaluation. Time-stamp each product-requirement compliance feature so that the rationale can be linked to the video record.

AFTER EACH DEMONSTRATION SESSION:

10. ACQUIRE MORE CUSTOMER REQUIREMENTS using different forms of the card sorting technique (lesson 2). One example is to ask stakeholders to grade the degree of compliance of products (cards) to requirements (categories). These grades can be quantitative (e.g. 0-7) or qualitative (good, average or poor fit). Also use triage sorting techniques described in lesson 1;
11. USE LADDERING TECHNIQUES to discover further important but non-discriminating customer requirements.

Figure 3.9: Part of the template for use during a supplier-driven product demonstration.

3.3.4 Template-3: user trial template

After product demonstrations in template-2, the evaluation team may recommend that the customer implement one or two products in the working environment for trial use for a limited period. The objective of this template is to guide the requirements engineering team to acquire customer requirements and product information sufficient to select or reject products as a result of customer-led product exploration. The template encourages the requirements engineers to explore product suitability in a more realistic environment. During this stage, the team looks, among other things, for the product's compatibility, integrability, or interoperability capabilities and that it fits into the organisation's existing system architecture without causing too much disruptions. The product is assessed for compliance with customer usability requirements, amount of time needed for training or how easy it is to learn and use the product. Also gathered at this stage is the information about how much tailoring, gluing, wrapping or bridging will be required in each product and the amount of bespoke elements to be developed. At the end of this process stage, template 3 guides the team to select one or both products for production use. Part of the template is shown in Figure 3.10 and the full template is described in Appendix 3f.

TO DO BEFORE THE PILOT PROJECT

1. Over a limited period, install the selected products in the user environment;
2. Design test cases to test the following: interoperability, integrability, usability, performance, reliability, learning curve and training.;
3. Work with main stakeholders to weight each category
4. Design a score sheet for allocating compliance scores;
5. Assemble an evaluation team composed of stakeholder representatives that will allocate scores during the duration of the pilot project. The team must have all the required technical skills as well as the application domain knowledge;
6. If possible negotiate to have a supplier representative on site during the duration of the pilot project to help with technical problems or have a dedicated contact person from the supplier.

TO DO DURINGTHE PILOT PROJECT

7. Each evaluation team member allocates scores on the interoperability, integrability, usability, performance, reliability and the learning curve of each product. For usability Nelson's Usability Heuristics can be used;
8. Record all decisions behind all scores;
9. Record all the problems experienced during this period including the quality of the supplier's response to technical queries, help desk and technical support;
10. Identify and acquire new requirements and required product features.

TO DO AFTER THE PILOT PROJECT

11. Collate all scores for each product into one final score;
12. Rank each product and select the preferred one;
13. Negotiate with the supplier to include the new features that were identified during the pilot project;
14. Negotiate contractual and legal issues with the supplier including licensing arrangement. The contract should spell out all the parties' rights and obligations..

Figure 3.10 Part of the template for use during user trials

Because of the iterative nature of the requirements acquisition and product selection each template might be used several times during a product selection process. The templates are applied iteratively in cycles of Acquire customer requirements and product information, Analyse acquired requirements and product information, Decide and Reject and are linked into the generic process model that is shown in figure 4.2. Each iteration reduces the number of candidate products as shown in figure 3.7. Different iterations may vary the set of features being assessed, the individuals making assessment, techniques or the evaluation criteria to be used.

3.4 The PORE process model

The reported studies reveal deficiencies in current requirements engineering processes for COTS product selection. It is surprising that similar studies have not been reported. Their value is clear. Empirical studies can reveal little-known requirements for new techniques, methods and tools. Indeed, more studies of current work practices might see more solutions which meet the real needs of requirements engineers. The 21 hours of elicited information reported in section 3.2 aim to redress the imbalance a little and are an important source of empirical data about current processes and problems, and a guide for future method development.

The results enabled the population of W-level processes of the PORE process model. The lack of guidance in current requirements engineering methods and COTS product procurement means that a model such as PORE can provide support for real-world processes. PORE is being developed using NATURE's process modelling language (e.g. Grosz et al. 1996). This language is flexible and enables description of both planned and unplanned processes. PORE is composed of a set of contexts. Each context is an association of a situation to a decision which might be taken and a process to undertake. Situations refer to the current state of the requirement (e.g. the ITT is complete) or the procurement process (e.g. supplier selection is finished). Processes are linked to these contexts to guide their use throughout the process. Dividing PORE into a set of situated W-level and A-level processes makes it potentially less prescriptive and more flexible and usable.

The PORE model was populated with processes elicited during the studies and designed to overcome elicited problems reported in sections 3.3 and 3.5.2. First there is a clear need to improve the process of acquiring requirements (e.g. problem P2.1). One possible solution is to use templates to guide acquisition process. Each template is, in essence, a frame with labeled slots to be filled by users. The templates are used to define both requirements for the software package and the degree-of-fit of each candidate product to these requirements. The template slots also enable the definition of dependencies between requirements (e.g. Dobson & Strens 1994). These dependencies are critical to an effective decision making during supplier and software package selection.

The requirements templates also improve the use of multi-criteria decision analysis (MCDA) techniques for supplier and software package selection (see problem P4.3). Although MCDA techniques have been used in software product evaluation methods (e.g. Kontio 1996), these methods do not provide guidance on how to apply them. The PORE templates overcome these limitations through guided requirements acquisition and product selection. The PORE method process provides W-level process guidance for technique use as well as A-level techniques such as 'requirement templates' linked to MCDM decision support software tools (e.g. AHP).

3.5 Summary and chapter conclusions.

The experiences and problems that are reported in this chapter indicate the need for new process guidance that is not covered in the existing methods. As COTS-based systems development becomes more widespread, stakeholders are more likely to express customer requirements in the form of what product capabilities are currently available in the market. Software products, and indeed software components will, provide the basis for a lingua franca for communicating a large number of implicit customer requirements already operationalised in off-the-shelf software products. As a consequence, requirement specifications need to be sufficient to enable effective product selection rather than complete with respect to the user's needs. This, in the opinion of this thesis, provides one of the greatest challenges for software engineering researchers and vendors in the near future.

The studies results show that requirements engineering for COTS-based system development have problems that are seldom experienced in the traditional system development. The first study identified 29 major problems that are not addressed by current requirements engineering and COTS-based development methods. The second study identified similar problems of which eleven were reported. The study identified a range of problems about the nature of requirements acquisition for COTS-based systems development that are not experienced in traditional development. Evidence has been found to support hypothesis H1.

To improve the situation, new methods, techniques and tools and guidance for acquiring requirements for COTS product evaluation and selection are needed. PORE

is one such method. However, since the problems identified in this chapter are very large and the complete PORE process is too long, the remainder of this thesis concentrates on the two iterative processes of requirements acquisition and product selection. The rational for choosing to concentrate only on these two processes is that compared with other processes, there is very little theoretical understanding of requirements acquisition for COTS product evaluation and selection. This thesis aims to fill this gap! This part of the PORE approach in which this thesis concentrates, has three main components that are fully described in chapter 4:

- a process model that identifies 3 essential goals that should be achieved by any COTS-Based Development process and prescribes four generic processes to achieve each of these goals as well as guidance and sequence in which these goals should be achieved
- a method box that includes methods, techniques and tools that are available to undertake and achieve each of the process goals
- a product model, requirement model and compliance model that provide semantics and syntax for modeling software products, requirements and compliance mapping.

These three components are integrated into an approach that provides a requirements engineering team with a coherent process guidance for an iterative COTS-based development process. In the next chapter, a theory of interleaved requirements acquisition and COTS software product selection is described to provide a theoretical basis for process guidance.

Chapter 4

Interleaved Requirements Acquisition and COTS Software Product Selection

This chapter describes the need for process guidance for COTS-based development process and proposes techniques and models for guiding the requirements engineering team during requirements acquisition and product evaluation. The chapter concludes by recommending a software tool.

Chapter 4:

Interleaved Requirements Acquisition and COTS Software Product Selection

A central contribution of this thesis is that the iterative process model for COTS-based systems development that this research predicts will improve requirements acquisition and product selection beyond the observed current practices reported in chapter 3:

- **H2** It is possible to design more effective methods, which directly address current problems in requirements engineering research.

In this chapter, section 4.1 describes the PORE's iterative process which interleaves requirements acquisition and product selection. Section 4.2 describes goal-based process guidance. Section 4.3 describes a multi-layered process guidance that identifies 3 levels of guidance. Section 4.4 describes models for guiding the PORE process. Section 4.5 describes PORE's method box. The method box includes methods, techniques and tools that help undertake the PORE process. Section 4.6 describes process situation rules that help infer current process situations based on the state of the compliance model. Section 4.7 describes the PORE process chunks which link the process situations, models and rules to provide situated process guidance. Section 4.8 provides summary and chapter conclusion.

4.1 PORE's iterative process

At the heart of PORE method is the iterative process of parallel requirements acquisition and product evaluation/selection. The iterative process is depicted in figure 4.1.

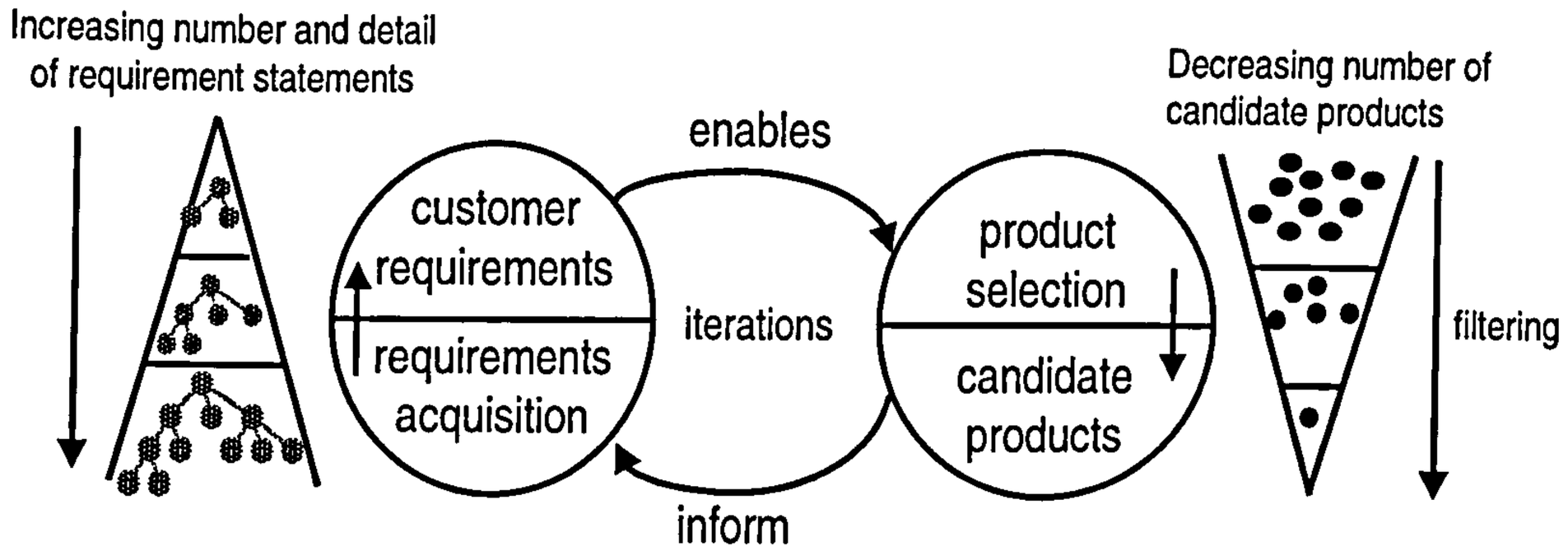


Figure 4.1. Overview of the PORE's iterative process of requirements acquisition and COTS product selection. Customer requirements enable COTS product selection and candidate COTS products inform requirements acquisition in small iterations.

The iterative process enables the team to reject COTS software products that do not meet core customer requirements. For example, at the beginning of the process, there could be many products and few customer requirements. As the selection process proceeds, the number of products is reduced and the number of customer requirements increases with customer requirements enabling product evaluation and product evaluation informing requirements acquisition.

The process model depicted in Figure 4.1 is the central component of the PORE approach. The process model is part goal-driven and part context-driven. The goal-driven part identifies critical, unavoidable decisions that have to be made about product selection at key points in the process. It prescribes processes to achieve these decisions or goals in a predetermined sequence. In contrast, the context-driven part reflects the realisation that it is difficult to prescribe sequences of lower-level processes to achieve higher-level goals, let alone which are the best techniques to use to achieve them. For example, information about customer requirements, software products, suppliers and procurement contracts is often not available to the evaluation team in the order in which it is needed, so the sequence of the acquisition, analysis and decision-making processes cannot be pre-determined. Furthermore, requirements acquisition and product selection processes are often performed simultaneously, in that the successful completion of one process often depends on the successful completion of the other. For this, PORE guides the requirements engineering team using information about the current context, or 'situation'. These situations are modelled using properties inferred from models of the customer requirements,

software products and compliance relations between requirements and product features defined by the requirements engineering team.

One of the main features of PORE's iterative approach is that it encourages the requirements engineering team to acquire, describe and analyse customer requirements at the same time as acquiring, modelling and analysing the candidate COTS software product. Advantages of the approach are two-fold. Firstly, acquired requirements enable COTS software selection and secondly, short-listed COTS software products can inform subsequent requirements acquisition to aid further software selection. This results in concurrent requirements acquisition and product evaluation with the processes of requirements acquisition and product evaluation performed in small simultaneous iterations. To provide process guidance for selecting/rejecting products, PORE provides three essential goals and 5 generic processes that must be performed in an iterative sequence to achieve each goal. The following section describes the 3 essential goals.

4.2 Goal-based process guidance

PORE defines 3 essential goals to select or reject candidate products according to compliance with:

- atomic customer requirements, (Goal 1);
- complex non-atomic requirements, (Goal 2);
- non-functional requirements such as architectural, reliability or usability requirements, (Goal 3).

The requirements engineering team should achieve these goals in a sequence. The sequence is designed to take account of real-world constraints such as the time needed to achieve each goal, and the availability of software product information at each stage. Atomic customer requirements (Goal 1) such as functional requirements are used in earlier stages of the process stages because its easy to determine their measurable fit criteria and to test for the presence or otherwise of a product's functional feature, (either the feature is present or not). In contrast, complex non-atomic functional requirements (Goal 2) and non-functional requirements (e.g. architecture and usability requirements, Goal 3) are used to evaluate a small number of short-listed products later in the process due to the complex and time-consuming nature of the compliance evaluation.

To achieve each of the 3 essential goals, PORE prescribes 5 generic processes which are essential to undertaking the iterative process:

(1) identify candidate COTS products – this process is essential for identifying candidates COTS products that are available in the market using guidance provided in Template 1 and recommended techniques such as the internet, market surveys or trade shows;
(2) acquire information about customer requirements, software products, suppliers and procurement contracts from stakeholders – the process is essential for acquiring system requirements from main stakeholders and information about products and their suppliers that were identified in process 1;
(3) analyse acquired information – once the team has acquired information from stakeholders, they then analyse it for completeness and correctness before making critical decision;
(4) use decision-making techniques to analyse and determine product-requirement compliance – once the team has analysed the acquired information for completeness and correctness, this process determines the degree of product-requirement compliance using Multi-Criteria Decision Making (MCDM) techniques;
(5) reject one or more candidate products that are non-compliant with customer requirements- this process rejects those product that have been found not to comply with the customer's requirements as defined by the essential process goal.

Figure 4.2 shows a route-map based on the notation in Assar et al. 1999 that links the 3 essential goals and the five generic processes. The 5 generic processes are also integrated and linked to the 3 PORE templates.

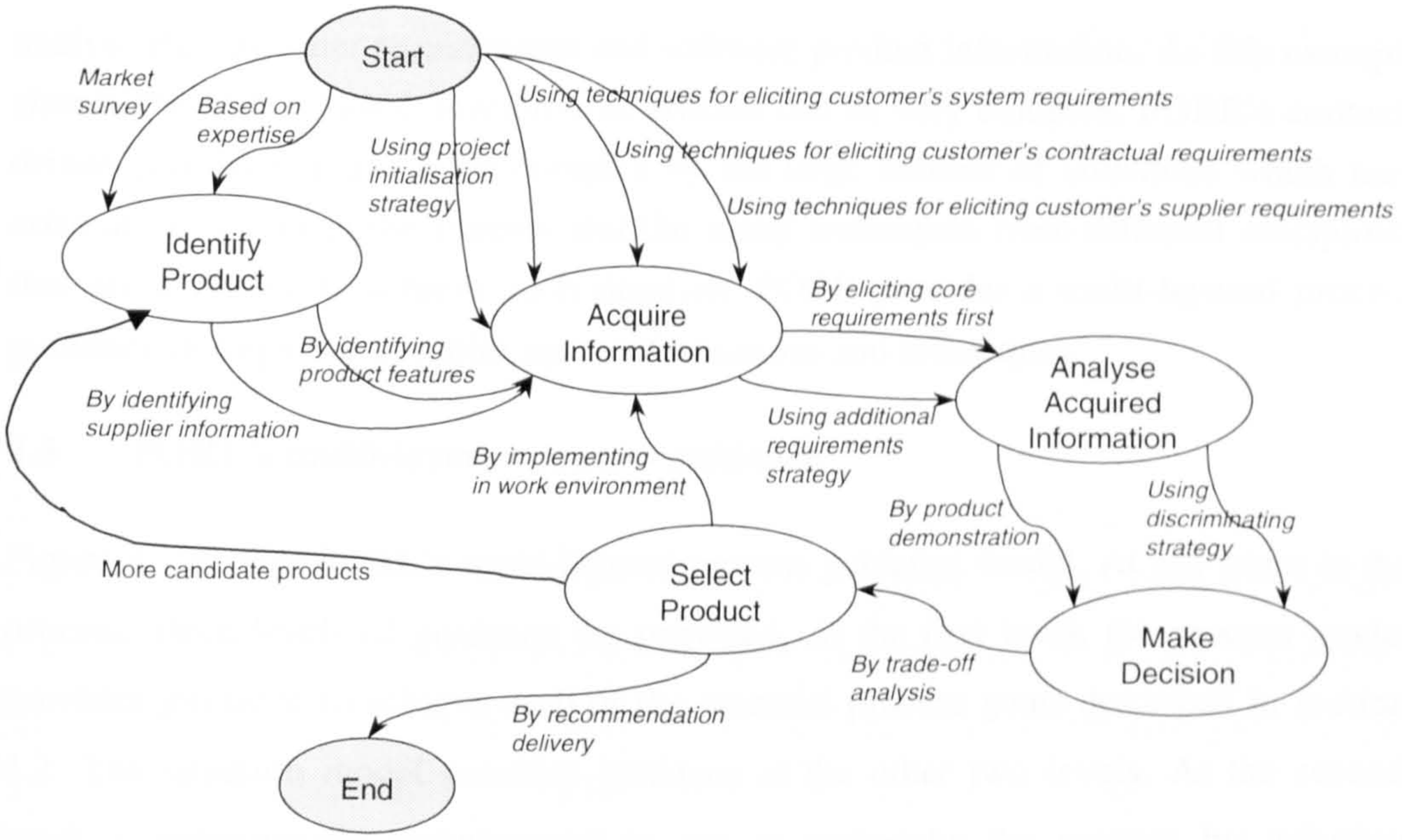


Figure 4.2: Graphical depiction of a route map showing PORE's 5 generic processes. The achievement of each essential goal is a broad sequence, in which the first processes of identifying of candidate COTS products and acquisition of information from stakeholders can be performed in parallel. The last process is the selection of one or more candidate products. Each process can be repeated many times.

The order in which the five processes are undertaken is context-driven, that is determined by the current process situation. PORE's *'situations'* are defined as in Suchman (1987) which states that *"every course of action depends in essential ways upon its material circumstances"*. For example, the first process is to acquire information from stakeholders and assume that the current situation is that there is no information from stakeholders (i.e. product information and customer requirements). Likewise, the last process of a successful process is the selection of one or more candidate products. This means that the current process situation is that complete and correct stakeholder information is available to enable decision-making. However, the sequence of the intervening processes is not predetermined, and each process can be repeated many times.

Furthermore, the 'current situation' restricts the sequences of these processes that are permissible. For example, if the "analyse acquired information" process reveals that there is insufficient information to make decisions about product-requirement compliance, then the team is advised to acquire more information. If the team is unable to discriminate between candidate products, then it is advised to further

analyse the customer requirements and software product information. As this example shows, the COTS-based development process can be very complex. PORE's context-driven process is made more complex by the large number of situations which may arise at any point in the process and the many techniques from different disciplines that are available to achieve each situation. PORE provides a multi-layered process guidance through this complex space of situations and techniques.

4.3 PORE's multi-layered process guidance

Figure 4.3 depicts PORE's multi-layered process guidance model. At any point in the process, three levels of guidance are provided. At the first level, the process model provides guidance to achieve each of the essential process goals described in section 4.2. The situation model provides guidance at the other two levels. At the second level, it recommends techniques(s) to use to undertake the process by inferring general properties about the requirements, product and compliance sub-models. At the third level, it recommends the content focus, i.e. 'current situation', for applying each technique based on inferences about the current contents of the requirements, product and compliance sub-models.

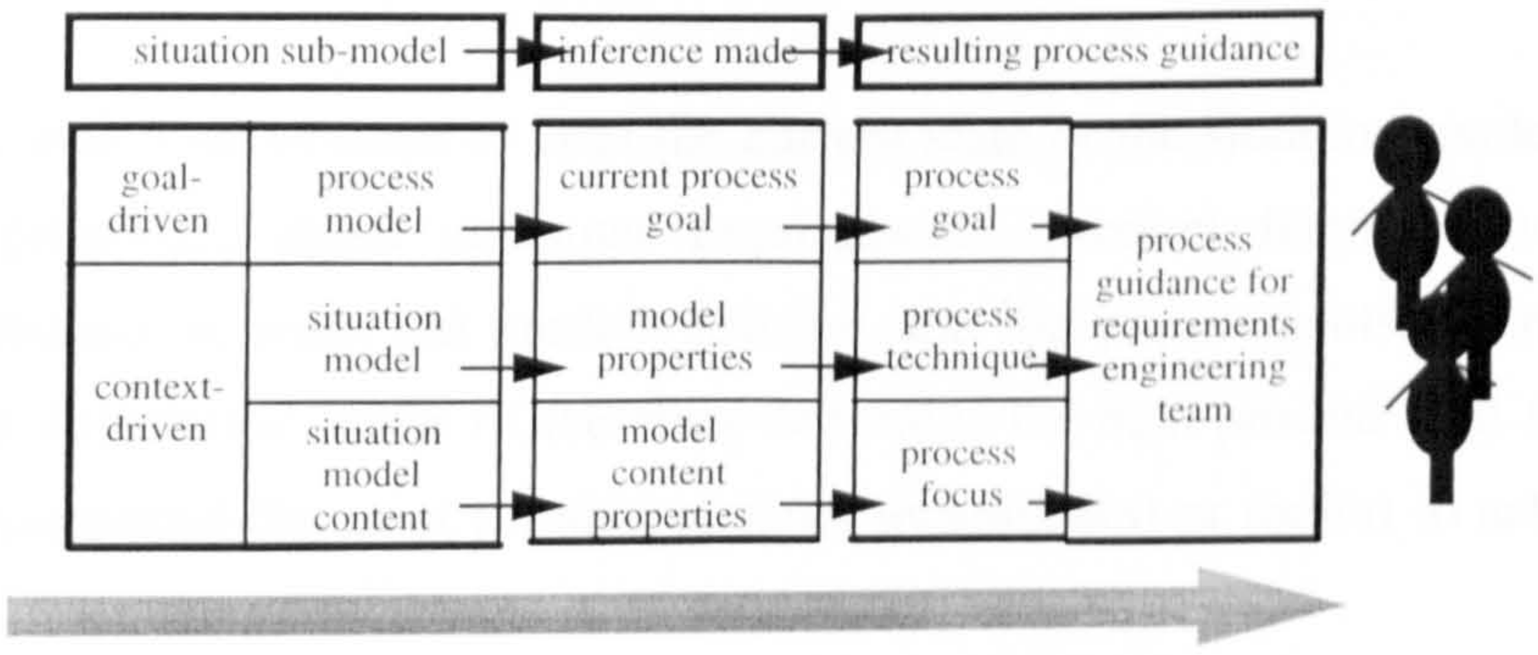


Figure 4.3: The three levels of process guidance that form the PORE process triplet. The current process goal is inferred from the process model. The technique to achieve this process is inferred from properties of the situation sub-model which is composed of the requirements sub-model, the product sub-model and the compliance sub-model. The focus of this technique's application is inferred from properties of the situation model content.

This multi-layered process guidance is given to the requirements engineering team in the form of a triplet:

{process-goal, situation, techniques-to-use}

The goal-driven process model described in section 4.2 specifies the process-goal part of the triplet, the context-driven processes specify the techniques-to-use and the situations for which the techniques are applied are inferred from the properties of the situation model. The definition of the current process triplet changes as new information is added to the sub-models and new inferences about the properties of the situation model are made. Of the three triplet elements, the process-goal part changes least and the situation changes most during each instance of a process.

The process defines a large number of possible 'situations' that are possible at any point. In addition, a large number of processes and techniques to be used in a single situation can sometimes be recommended.

4.3.1 Situation-based guidance

PORE uses a set of rules to infer the current state of the situation model. These rules infer properties about customer requirements, product features and compliance relationships between the product features and the requirements. From the inferred current situation(s), other PORE rules determine the next process goal to be achieved and recommend the most suitable method, technique(s) or tool(s) to achieve the goal chosen from the method box.

To demonstrate the importance of situated process guidance, consider two simple example situations:

Situation-1: if the requirements sub-model contains a number of requirements which are all compliant with the product features of several products, then the PORE process model advises the team to acquire more customer requirements which enable more effective discrimination between products. The PORE method box then recommends techniques such as card sorts that are more effective for acquiring such discriminating requirements.

Situation-2: if the requirements sub-model contains a small number of behavioural requirements, then the process guidance advises the team to acquire more behavioural requirements, and the method box recommends techniques such as use case analysis and user walkthroughs of product demonstration copies.

In both examples, effective technique selection is determined by both the process goals and the situations inferred from the properties of current state of the requirements, product and compliance sub-models.

Two questions that arose during design of PORE were (i) how to model product-requirement compliance? and, (ii) what features of software products and attributes of customer requirement attributes to model in order to define the situations that can effectively guide the PORE process? The solution is 3 sub-models - the product sub-model, the requirements sub-model and the product-requirement compliance sub-model - which are at the heart of the PORE method. All 3 sub-models, when combined, provide a model of the current situation (i.e. the situation model). The following sections define and describe each sub-model.

4.4 Models for guiding the PORE process

This section describes the 3 sub-models. Section 4.4.1 presents the software model and its meta-concepts, section 4.4.2 presents the requirements model and its attributes, and section 4.4.3 presents the product-requirement compliance model together with compliance mappings. The 3 sub-models are modelled using an existing modelling technique, the Unified Modelling Language (UML) (Rumbaugh et al. 1998).

4.4.1 Product model

During product selection, not all product information is available to the team, so the PORE approach is pragmatic and encourages the team to first model observable rather than non-observable features of the product. PORE also encourages to model product features that are directly acquired from suppliers through interviews, questionnaires and other acquisition techniques. These real-world limitations are an important constraint on the design of the product sub-model to ensure that it is both usable and useful.

To enable effective COTS product evaluation and selection, there is a need to model critical features of software products in three parts:

- the product model models the observable behaviour of the product, and in particular, how the user interacts with the product.
- the product model also models the product's articulated goals using goal-based requirements methods (e.g. Anton 1997);
- it also models the product's architecture using architecture modelling techniques such as those reported in Shaw (1996), Garlan et al. (1995) and SEI (1998).

Figure 4.4 depicts the software product meta-model.

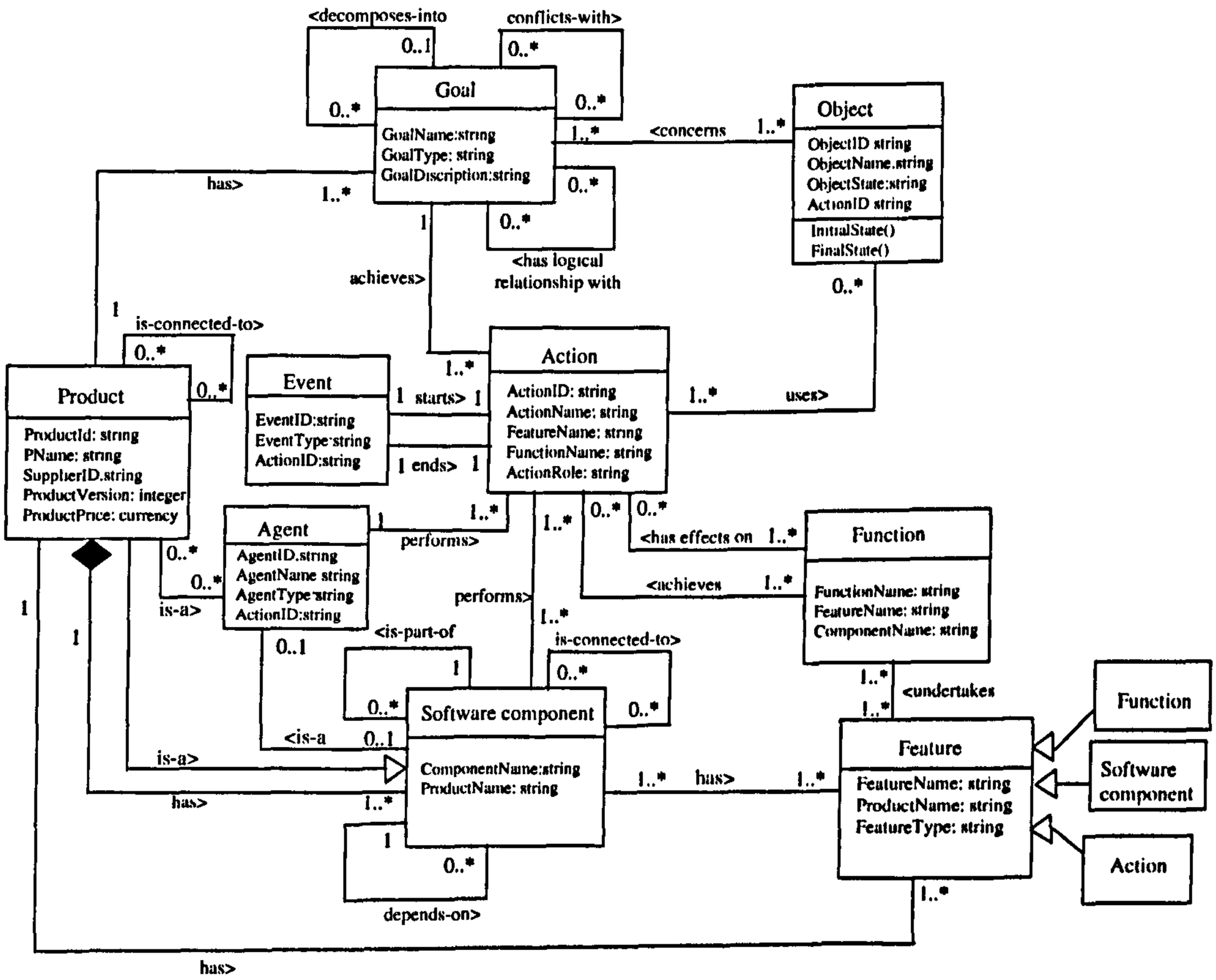


Figure 4.4: The PORE's software product meta-model and its primitive concepts and the meta-relationships linking the meta-concepts.

To enable a requirements engineering team to model the properties of the software product, the PORE approach uses modelling concepts such as *goals* to be achieved, *objects* to be used, *actions* taking place, *events*, *agents* to perform actions, *components* involved in actions, *functions* to achieve actions, *features* to undertake functions and

relationships between meta- concepts (Sutcliffe et al. 1998). These meta-concepts are instantiated during the requirement-product compliance mapping process. The purpose of the meta-model is to model different situations as the instances of the meta-model in order to enable situated requirements acquisition and product evaluation processes by the inference of requirements, product and compliance sub-model properties.

The primitive concepts of the product model are described:

- a **goal** is a high-level objective that the system should meet (Darimont & van Lamsweerde 1996). Goals are achieved by actions performed by agents. The goals are decomposed into alternative combinations or logical sub-groupings and may sometimes conflict with each other (Anton & Potts 1998). A typical example of a goal for a requirements management tool is to '*manage requirements documents*';

- an **action** is a process linked to the attainment of a goal. Each action can be cognitive, physical, system-driven or communicative. Each action can involve one or more agents, use one or more objects and may result in a state transition which may change the state of the object. Actions are linked to each other using action-link rules (Maiden et al. 1998). In a requirements management tool, an example of an action is *Create* requirement or *Copy* requirement;

- an **agent** is a type of object which performs or processes actions (Darimont & van Lamsweerde 1996). Each agent has certain features that determine its capabilities to perform the desired actions and are responsible for completing and/or satisfying goals through performance of actions. Each agent is either a human agent or a software component, thus enabling the requirements engineering team to model the system boundaries (degree of automation) of each product. A *word processor* is an example of an agent in a requirements management tool;

- an **object** is something of interest in the domain. Object instances can evolve from state to state through the application of actions. Each state of an object at some time is defined as a mapping from an object to the set of values at that time of all

features of the object. Objects are modelled in use cases to describe both the product's information features and the customer's information requirements. A *requirements document* is an example of an object;

- **functions** are a mode of action by which the product fulfils its purpose. They are the services and capabilities provided by the product and specify what the product is capable of performing. Functions define the behaviour of the product and the fundamental processes or transformations that the product and the hardware components of the system perform on inputs to produce outputs. The behaviour of a product's function can be mathematically characterised as a function that receives some input x and produces some output y. An example of a function is the 'function *Filter*' that a requirements management tool can use so that only those requirements that match the user's specified criteria are displayed;

- a **software component** is an independently deliverable set of software services available to users or to other components (Brown & Short 1997). A software product itself can also be a component. Components interact or collaborate with each other through connectors to accomplish solutions or to undertake complex functions. The structural features of components collectively form the component's architecture (Shaw 1996, Garlan et al 1995). A *database*, *word processor*, *configuration tool* and *version management* tool are components of a typical requirements management tool.

- a **product feature** is distinctive or characteristic element of a software product or component. Product features are characterised or specialized as functions, actions or software components. Typical examples of features of an e-mail product include Address book, Dictionary, Auto reminder, Auto spell check, Sending/Receiving new mail, etc.

In addition to the meta-concepts, relationship types between meta-concepts are also specified:

- **connectors** facilitate interaction between products or components of a product in order to form executable structures. Examples of connectors include protocols, procedure calls, remote procedure calls, buffers, event broadcasts, constructs, etc. Connectors are further divided into types such as data-carrying connectors, control-oriented connectors and hybrid forms that exhibit both characteristics to some degree;

- **dependencies**: the product sub-model defines dependence relationships between components. One component X is dependent on another component Y if component X contains a call to component Y, that is X is a dependent of Y. In its simplest form, dependence relationship is a link between X and Y indicating that X depends on Y to achieve its goal. This form of dependence may result in a sequence of achieving goals and may remain static, i.e. remaining the same over the lifetime of the products or may be dynamic, existing only when required or demanded. The dependence relationship needs to be modelled in order to determine scope of the product being evaluated and to avoid adverse effects such, for example, if Y fails to achieve its goal, X will be adversely affected since it depends on it. The type of freedom allowed between X and Y determines dependence types.

Ray (1996) and Kaasboll & Motschning (1996) identify the following types of dependency relationships that may exist between X and Y:

- functional dependencies in which the correct behaviour of one component requires the correct operation of another component (e.g. functional dependencies between the word processor and the database);
- trigger dependencies that define timing associations between two components (e.g. between the configuration manager and the version control manager);
- precedence dependencies in which one component has to complete its operations before another one starts its operation;
- constraint dependencies that constrain or restrict the behaviour of one component by another component;

- some other dependence relationships are linking, i.e. X is linked to Y; Import and Export, i.e. X imports or export data from and to Y; Integration, i.e. X integrates with Y to allow bi-directional exchange of data through interfaces.

These dependency relationships, in combination with component connectors, provide a simple but useful basis for modelling COTS software products as exemplified in Figure 4.5:

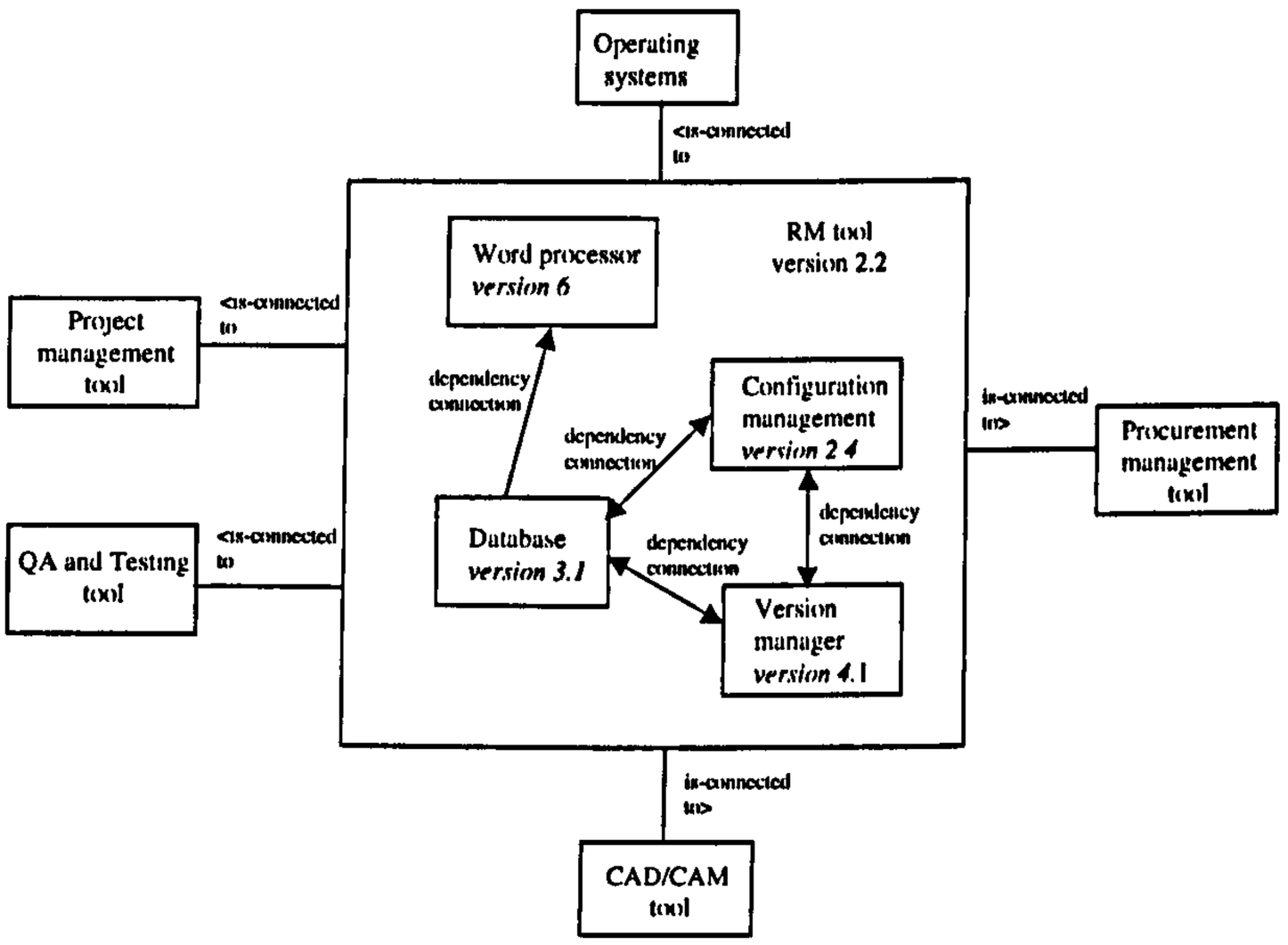


Figure 4.5 An example of an instantiation of the product meta-model showing requirements management tool's components and the product's connection to other products and the connection between components and the dependency relationships. The boxes represent other products or components. The components determine the scope of the product but they are all independently developed as indicated by their different version numbers. The figure also show that the requirements management tool can be linked to other software products via different connectors described above and that the requirements engineering tool itself evolves independently of its components and the products that it is linked to.

The major consequence of dependency relationships is that although component X and component Y may co-exist and are tightly coupled to each other, they can change and evolve individually and independently, at different speeds and in separate non-

synchronised life-cycles. A typical example is the components of a requirements management tool. Most requirements management tools have a *database* to store requirements, a front-end *word processor* to enter requirements into the database, a *configuration management tool* and *version manager* to control and manage requirements changes. Each of these components have independent evolution cycles that are not dependent on the existence of the other components since they are more likely to be developed by different suppliers with different development strategies and directions. This is a source of risks and the scope of the dependence needs to be identified during requirements acquisition and product evaluation.

4.4.1.1 Rationale for a software product model

PORE's software product sub-model enables the evaluation team to model the complexities of COTS software products. Software products often interact with users during tasks (e.g. updating a requirement statement), have internal system functions (e.g. checking requirement compliance with a standard) and have increasingly complex system architectures to support user tasks and internal system functions. Modelling techniques such as task modelling from human-computer interaction (e.g. Johnson 1992), functional modelling from software engineering (e.g. Dardenne & van Lamsweerde 1993) and architecture modelling from system design (e.g. Garlan 1995) are all drawn on to model a software product at these three levels, (i.e. behaviour, functional and architecture levels).

The product's architecture provides a description of the components and the complex interrelationships between the components (Garlan 1995). Components are the building blocks of the product or system and the architecture is the topological interconnections of the components and is concerned with how components interact, co-ordinate, co-operate and communicate with other components. From this architectural lens, a software product can be viewed abstractly as a configuration of the components and connectors. The components are connected in a way that enables the system to meet its requirements. The selection of the components and interfaces has a big impact on how the original requirements will be met.

Product architectures are concerned with both the functional and non-functional requirements/attributes of the system. The non-functional attributes are mainly concerned with the ability of the product to integrate with other products at application level while the functional attributes are concerned with the data level integration. The product's architecture helps to reason about some architecture properties such as physical distribution of components, process communication, and synchronisation between components and processes. Some other examples of architectural properties that are critical in a COTS-based development process that involves integration of many products are flexibility, reliability, portability, openness, data interchange, migration, standards, platform issues, functional, data management, security, user development, interoperability (e.g. Brown 1998; Carney 1998; Voas 1998).

Furthermore, there are strong dependencies between the product's behaviour properties, its functions and its architecture, and these dependencies need to be modelled in order for the requirements engineering or evaluation team to make effective decisions about product-requirement compliance. For example the observable behaviour of the product largely depends on its functions; functions determine operations to be performed; operations determine observable properties and the displayed results. The product achieves its functional goals through its structural or architectural and basic properties that act as glue between the product, its operational environment and the hardware components of the enterprise system. Also stakeholders often express the requirements for their systems in terms of its behaviour, functions and architecture, therefore it makes it easy to determine product-requirement compliance if software products are modelled using the same constructs as customer requirements. As a consequence, PORE's product model enables each software product to be modelled in these three different ways to improve the effectiveness of compliance checking.

4.4.2 The requirement model

A requirement is a capability that a software system must supply or a quality that a system must possess in order to solve or achieve an objective within the system's conceptual domain. It is a '*measurable statement of intent about something that the product or system must do or a property that a product must have or constraint on the system*', (Robertson & Robertson 1999). A critical factor in successful acquisition of

requirements is to understand not only what the system under consideration should do (functional requirements), but also the way in which it should provide its services (non-functional requirements). A broader view of requirements acquisition, therefore, goes beyond the description of what the system is expected to do (system's functionality) and include system properties and constraints under which the system must operate (non-functional requirements). In the COTS-based development process, this view is taken even further to include information about product suppliers such as the supplier's technical capabilities, application domain experience and ISO standard certification (supplier requirements) and legal issues involved in product procurement such as negotiating contract terms and conditions and licensing arrangements (contractual requirements). The PORE method uses the requirement sub-model to both acquire and elaborate the requirements statements and to check requirements-product compliance during the iterative process of requirement acquisition and product evaluation/selection. Figure 4.6 depicts the meta-concepts and meta-relationships of the requirement model.

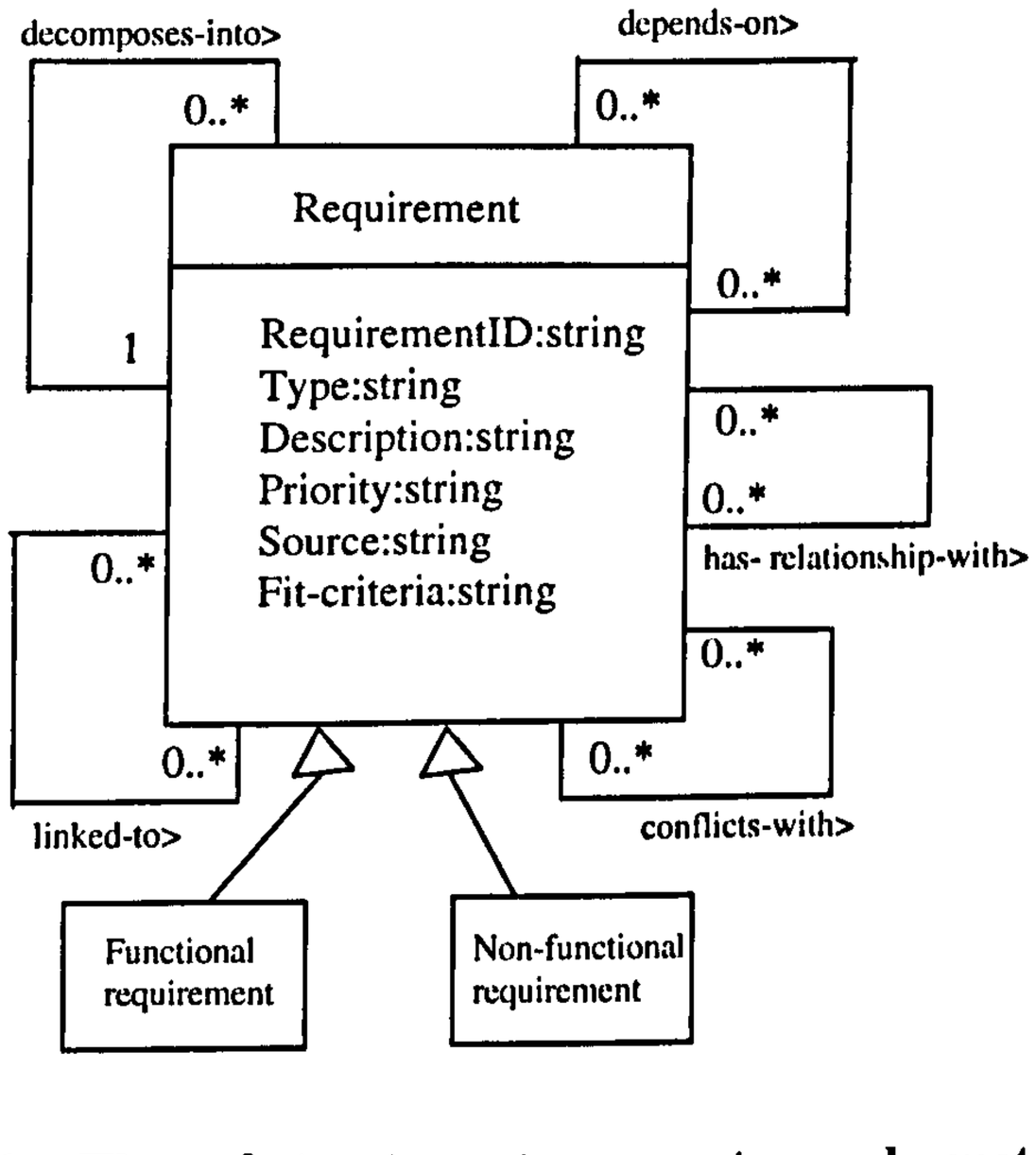


Figure 4.6: The abstract meta-concepts and meta-relationships of the requirement sub-model.

The following requirement meta-concepts are defined:

Requirement type is a special class of requirements having the same characteristics (Thayer & Dorfman 1997). Typical examples of requirements are functional, non-functional or global requirement types (Robertson & Robertson 1999):

- **functional requirements** are actions that the product must be able to take. They specify the purpose of the software or what it has to do to provide services to the users and the functions that the product must be capable of performing. They are the *'fundamental subject matter of the system'*;
- **non functional** requirements are the 'behavioural properties that the specified functions must have, such as performance or usability.
- **global requirement** types are '*requirements or constraints that apply to the product or system as a whole*', i.e. '*the purpose of the product or system or the customer of the system is a global requirement'*. Typical examples are:
  - **project constraints** that 'identify how the eventual product must fit into the world. For example the product might have to interface with or use some existing hardware, software or business practice, or it might have to fit within a defined budget or be ready by a defined date'.
  - **project drivers** that 'are the business- related forces. For example the purpose of the product is a project driver, as are all of the stakeholders - each for different reasons'.
  - **project issues** that 'define the conditions under which the project will be done'.

Requirement attributes are 'descriptive information associated' or attached to a requirement that provide specific important details and information about a requirement. An attribute has a ***label***, (i.e. the name of the attribute such as risk, priority, type, author, owner, ID number, version, status, revision number, description, fit criteria, etc.) and ***value*** (i.e. information assigned to the attribute label such as text or number, e.g. the value of the ***priority*** could be *Low*, *Medium* or *High*). Most attribute information is project-related and can help in planning, communicating and monitoring development activities through the development life-cycle.

Requirement measurable fit criteria (Robertson & Robertson 1999) enables the team to determine whether or not a solution satisfies the original requirement. The fit criteria are benchmarks, or goals that determine whether the eventual solution satisfies the requirement. Fit criteria are 'precise, quantified goals or testable statements of the requirement that contain numbers or measurements that the solution has to meet'. For functional requirements types, fit criteria 'are the yardstick that is used to test whether or not the function has been successfully carried out'. Fit criteria for non-functional requirements types 'quantify the necessary behaviour or quality of the system'. Fit criteria 'provide some quantified targets that when tested, will reveal the solution's degree of conformance with the requirement'. Each requirement has a fit criteria and the fit criteria depends on the action being required, (Robertson & Robertson, 1999). Requirements must also be verifiable, i.e. it must be possible to have some kind of verification that checks the end product and give a true or false binary answer (Stevens & Martin, 1998). The objective of verification is proof of non-conformance as efficiently as possible.

In addition to meta-concepts, there are many relationships identified between requirements. Requirement relationships maintain important linkages between requirements and from requirements to all development products that emanate downstream from them. The links makes it easy to ascertain the impact of any changes, and to determine the requirement status. Some of the identified relationships are:

- *decomposes-into* which maintains links between a parent high-level requirement and the lower-level detailed requirements which originated from this requirement;
- *linked-to* which determines all other requirements which are linked to this requirement;
- *conflicts-with* that identifies requirements which are in conflict with this requirement;
- *has-relationship-with* which identifies requirements that have association relationship with this requirement;
- *depends-on* which keep track of requirements that have an impact on other requirements or requirements that use the same information or have a change

effect on other requirements. Dependency relationships might exist where solution to a particular requirement has a positive or negative effect on solutions to other requirements. Cross-referencing requirements captures these dependency relationships. Some requirements, especially global constraint requirements, have an impact on all other requirements.

4.4.3 The compliance sub-model

A prerequisite for effective product selection is compliance between one or more features of each candidate product and one or more customer requirements. Compliance is defined as a mapping between a problem (i.e. a customer requirement) and a potential solution to that problem (i.e. a product feature). The compliance sub-model enables the mapping of customer requirements to product features and to check the degree of compliance between the requirement and the product feature. The product-requirement compliance sub-model provides an essential basis for technique selection and use. The compliance meta-model is depicted in Figure 4.7.

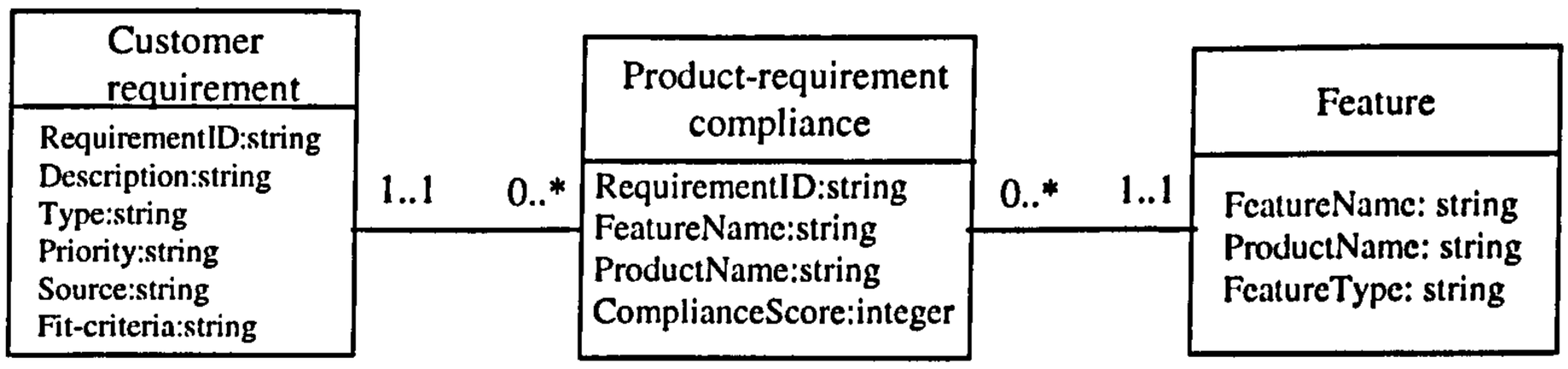


Figure 4.7: The structure of the compliance meta-model, and the relationships between the requirement, product and compliance sub-models.

Figure 4.7 depicts the relationship between the product sub-model and the requirements sub-model described earlier. The compliance sub-model models the degree of compliance between the customer requirements and the candidate COTS product. Compliance is modelled as a set of relationships between customer requirements and product features. Attributes on each relationship have values to indicate whether or not there is compliance. In order to be able to choose between product features, there is a need to measure the degree of compliance and confidence

between requirements and product features. A simple numerical measurement scale is used to determine the degree of confidence as shown in Table 4.1.

The goal of compliance mapping is to map customer functional requirements to a product's technical features based on the following mapping assumptions (Zaremski and Wing, 1996): associated with each product feature, PF, is a signature PF*sig* and a specification of its behavior, PF*spec*. Signatures describe a feature's type information and specifications describe the feature's dynamic behavior. Therefore given product feature, PF = (PF*sig*, PF*spec*) and requirement, R = (R*type*) the generic compliance mapping algorithm, Map is defined:

$$\text{Map: ProductFeature, Requirement} \rightarrow \text{Bool}$$
$$\text{Map (PF, R)} = \text{map (PF}sig\text{, R}type\text{)} \wedge \text{map (PF}spec\text{, R}type\text{)}$$

PF and R map iff the signature and specification of the product feature matches the requirement type. Table 4.1 shows an example of mapping between requirements and product features:

RequirementID	FeatureName	ProductName	ComplianceScore
R1	AddressBook	ExpressOutLook	4
R25.1	Calendar	Eudora	2
R20	AutoReminder	ExpressOutLook	5
R20	AutoReminder	Pine	0
R10.5			0
R5	SpellChecker	Communicator	3
R5	SpellChecker	ExpressOutLook	5

Table 4.1: An example of compliance mapping between requirements and product features. The figure shows that requirement R1 is mapped to product features *AddressBook* with a compliance score of 4, R20 is mapped to 2 product features with a compliance score of 5 for one product and 0 for the other. R10.5 is not mapped to any product feature. The degree of compliance indicates how strongly the product feature meets the customer requirement.

As mentioned earlier, the requirements acquisition and product selection depend on the successful completion of other processes. At any point in the iterative process, there are mappings between requirements and product features. The mappings provide the evaluation team with the context or situations that guide them about what to do

next. These situations are modelled using inferred properties of the current sub-model of the customer requirements, software products and compliance relationships defined by the evaluation team and provide situated process guidance described in section 4.3.1.

The three sub-models described above are integrated into hybrid situation-driven approach that provides a requirements engineering team with a coherent context-driven process guidance. However, PORE's context-driven process guidance is made more complex by the large number of techniques from different disciplines that are available to achieve each process goal. PORE identifies numerous techniques, methods and tools to acquire and analyse information, and to make complex decisions and select candidate products and makes them available through its method box. For each technique, method and tool, where available, PORE gives information sources about the technique, advice about the technique's use and where possible, prototypical examples of its use. The following section describes the PORE method box.

4.5 PORE's methods box

COTS-based development is a multi-disciplinary paradigm that requires techniques, methods and tools from many disciplines. This is seldom the case with the traditional systems development paradigms. PORE's method box includes methods, techniques and tools that are available to help undertake and achieve each of the situations and processes. PORE templates (described in chapter 3) provide guidance for selecting the most suitable method, technique or tool from the method box. Some of the techniques, methods and tools available in the PORE method box and integrated within the iterative PORE process are:

- knowledge engineering techniques such as card sorting and laddering (e.g. Rugg and McGeorge 1995) which are useful when acquiring information about categories of products, suppliers, contracts and hierarchical information about product properties and customer requirements;
- feature analysis techniques (Kitchenham & Jones 1997) for aiding when scoring the compliance of each product feature to each customer requirement;

- MCDM (Multi-Criteria Decision Making) techniques such as AHP (e.g. Saaty 1990) and the out ranking method (e.g. Fenton 1994) for aiding in the decision making process during the complex product ranking and selection process;
- COTS-based development methods such as OTSO (e.g Kontio 1995) for product evaluation and selection, CISD (e.g. Tran et al. 1997) and IIDA (e.g. Fox et al. 1997) for product identification and integration and IusWare (e.g. Moriso et al. 1997) for decision analysis.
- requirements engineering methods such as Volere (e.g. Robertson 1997) for aiding the requirements engineering process and equirements acquisition techniques such as ACRE (e.g. Maiden and Rugg, 1996) for acquiring customer requirements;
- product (or component) identification tools such as the internet or Agora (e.g. Robert et al. 1998) for identifying products or components available in the market;
- methods such as ATA (Architecture Trade-off Analysis) for analysing architectures and SAAM (Software Architecture Analysis Method, SEI 1998) for evaluating software product architectures.

The above list is not exhaustive. The PORE approach is designed to allow additions of relevant techniques as and when they become available in the market place. The techniques are divided into groups that deal with specific situations during the process. As well as integrating these techniques, PORE also provides rules that infer the current process goal and situation and, therefore, the technique to help solve the situation and meet the process goal. The following section describes such process situation rules.

4.6 Process situation rules

The situation rules infer the current situation by inferring properties about the current state of the compliance sub-model from attribute values of requirements, product features and product-requirement compliance relationships. Different situation rules infer process goals, different properties of the situation model and properties about the

semantic contents of the requirement, product and compliance sub-models. The inference is made possible using the meta-schema. Several rules infer properties about the contents of the product sub-model, and in particular, missing goals, actions, functions and components from a software product sub-model, so that a compliance check to a behavioural or functional requirement can be made based on complete information. The situation rules are defined in the form of a logical implication, *IF <condition> THEN <action>*, that are expressed in Visual Basic implementation statement as shown in the examples below and in more detail in chapter 5. Complex rules that infer properties about the collective contents of the requirement, product and compliance sub-models are defined. For example, one undesirable property is that there are compliance relationships between all product features in the product sub-model and all requirements in the requirement sub-model. This means that there are no effective discrimination requirements or product features. This is expressed by the following rule as a Visual Basic code:

```
If lisProduct.ListCount > 1 Then
   Call produceAdvice("NonDescriminatingRequirements", "Insufficient
   descriminating requirements or product features")
```

Rules that infer model content properties are specified to determine the current 'situation' of the process guidance, that is to infer the *situation* part of process triplet described in section 4.3. The rules were derived from the interviews undertaken with experienced software engineers reported in chapter 3 as well as from basic research.

The PORE rules infer situations that trigger process chunks (see Appendices 4a & 4b). For example, in the process chunk 1.5 given in section 4.6, the rule that detects that there are insufficient requirements to enable product selection (where insufficient is defined to be below a predefined threshold number of requirements) and therefore infers the situation *insufficient(requirements)* is expressed as below:

```
ElseIf RequireVal < RequirementInsufficient Then
    Call produceAdvice("InsufficientRequirements", "Insufficient
    requirements")
End If
```

Different types of rule sets infer properties about the relationship values of the compliance sub-model from attribute values of requirements and product features.

The rules infer compliance sub-model properties to guide the process chunks defined above to achieve their goals. So far the rules that have been specified infer ***situations*** that guide the trigger of process chunks (see Appendix 4a) and the selection of techniques to achieve all of 5 PORE's generic processes (identify, acquire, analyse, make decision, select and also see Appendix 4c). Table 4.2 lists an example of such situations. A complete list of situations that can arise during product selection is given in Appendix 4d. The following section presents and discusses the PORE process chunks and how they combine all the PORE components to provide the requirements engineering team with effective process guidance.

Situation Name
Empty requirement model
Empty product model
Insufficient customer requirements
Insufficient candidate products
Insufficient product features
Insufficient supplier requirements
Insufficient contractual requirements
Insufficient architecture requirements
Insufficient behaviour requirements
Insufficient functional requirements
Non-discriminating requirements
Non-discriminating product features
No compliance mapping relationship
Small decision making space
All products rejected

Table 4.2 An example of some of the situations identified in PORE. A full list of the situation that can arise is given in Appendix 4d. The situations were identified through real-world COTS product selection case studies and through interviews with experienced systems developers.

4.7 PORE's process chunks

The definition of PORE's process chunk is based on the process view of the NATURE process modelling formalism (Rolland & Grosz 1994, Plihon & Rolland 1995). Each process transforms a product (e.g. a requirements model) from an initial situation into a result which is the target of the intention of the process chunk (e.g. an improved requirements model). PORE's situation model and rules are linked and

integrated through process chunks to provide situated process guidance. Each process is modelled as a collection of process chunks that are combined in different sequences to form different processes to achieve different goals (see Appendix 4e). Each PORE process chunk has 6 attributes:

```
Process-Chunk:
        Name: Unique-identifier
        Goal: {the goal to be achieved by the process chunk}
        Process: {generic processes for achieving the goal}
        Situation: {property (sub-model)}
        Input-information: {content(situation sub-models}
        Technique: {technique in PORE method box}
End-Process-Chunk
```

The process chunks are also combined and linked with the process triplet (see section 4.3.1) to provide multi-layered process guidance (see section 4.3). The goal of each process chunk is the process goal to be achieved through the application of the chunk. Each process itself is either goal-driven or context-driven, but not both. If it is goal-driven, the process defines one or more goals to undertake to achieve the process. If it is context-driven, the context is defined using one or more inferable properties of the current situation model. The input information is the content of the requirements, product and compliance sub-models that is manipulated in the process. The technique attribute defines one or more techniques that available in the PORE method box to achieve the process goal.

The high-level process-chunk given below illustrates the goal-driven process depicted in Figure 4.2.

```
Process-Chunk:
        Name: 1
        Goal: Reject COTS software non-compliant with goal
        Processes: Identify candidate COTS products and Acquire-information
        THEN
                Analyse-compliance THEN
                Determine-non-compliant-product THEN
                Reject-non-compliant-product
        Situation: None
        Input-information: None
        Technique: None
End-Process-Chunk
```

To complete each cycle of rejecting COTS software products that are non-compliant with some subset of customer requirements, the four processes in the process chunk

are undertaken in a strict sequence characterised by the 'THEN' statements in the chunk. However, the processes of identifying candidate COTS products and acquiring customer requirements can be done in parallel. As this chunk describes a high-level process, it has no triggering situations, no prerequisite information input, and no specific techniques to recommend.

To achieve the three essential process goals defined in section 4.2, four high-level process chunks are defined. In turn, each chunk defines process sub-goals to be achieved. To illustrate this with an example, the high-level process chunk to achieve the first essential goal, which is '*to reject products that are non-compliant with essential, atomic functional requirements*', is described below. The example is further illustrated using an example which aims is to reject e-mail products that are non-compliant with requirements '*the system shall enable the user to maintain a customised address book, and the system shall operate on Windows-NT and MacOS version 8.1*'.

To provide process guidance, the process chunk specifies 9 fine-grain process goals to be achieved before the higher-level process goal can be achieved. Each of the 9 sub-goals corresponds to one of the four coarse-grain processes defined in Figure 4.2, (i.e. acquire information, analyse acquired information, determine product-requirement compliance and reject one or more products). The process goals define the <goal-process> part of the process triplet. However, the order in which the goals are achieved is context or situation-dependent, that is, it depends on the properties inferred about the situation model and its contents. The example of the process chunk is given below:

```
Process-Chunk:
      Name: 1.1
      Goal: Reject software products non-compliant with essential-goal-1
      Processes:   Acquire customer-atomic-requirements
                   Acquire contractual-requirements
                   Acquire supplier-requirements
                   Acquire product-information
                   Analyse product-requirement-compliance
                   Analyse supplier-requirement-compliance
                   Analyse contractual-requirement-compliance
                   Determine non-compliant software products
                   Reject one or more non-compliant products
      Situation: None
      Input-information: None
      Technique: None
End-Process-Chunk
```

In turn, to achieve the goal 'acquire atomic customer requirements' a large number of situated processes are defined. One such situation, *empty(requirement-sub-model)* defines that the requirements model contains no requirements with which to check

compliance with each product. In this situation, PORE recommends the use of a range of techniques which include *interviewing, brainstorming* and *use case analysis* to acquire a first collection of atomic essential functional requirements for the customer's future system. The process chunk that achieves this situation is shown below:

```
Process-Chunk:
        Name: 1.2
        Goal: Acquire atomic-functional-requirements
        Processes: none
        Situation: empty (requirements sub-model)
        Input-information: none
        Techniques: {interview, use-case analysis, brainstorm}
End-Process-Chunk
```

At the same time, the process model also recommends the requirements engineering team to acquire product-information, product-contract information and supplier-information for all candidate products, for similar reasons. At the beginning of the process, the product sub-model contains no information about candidate products, that is the situation is *empty(product sub-model).* The process then recommends the use of techniques such as questionnaires, data analysis and other diverse information sources to acquire the essential product, supplier and contract information:

```
Process-Chunk:
        Name: 1.3
        Goal: Acquire product-information
        Processes: none
        Situation: empty(product sub-model)
        Input-information: none
        Techniques: {questionnaire, internet, data-analysis}
End-Process-Chunk
```

The situations for the above two process chunks (i.e. 1.2 & 1.3) are very simple to infer and the recommended techniques are obvious. However, other process chunks are more complex and lead to the recommendation of less obvious process guidance. For example, the next process chunk is linked to the situation *'no-discriminating(requirements)'*. This happens if the requirements sub-model contains insufficient number of customer requirements that enable the team to discriminate between candidate products. For example if the number of customer requirements that have product-requirements compliant mapping relationship is lower than a predetermined threshold. In this situation, the process model recommends that the team acquire more customer requirements which help to discriminate between products using the card sorts technique. Using the card sort technique, a member of the requirements engineering team writes all candidate product names on 3"x5" cards and asks stakeholders to use the cards to sort the products into categories. Criteria for sorting such as *"product compatible with Microsoft™ Word"* indicate customer requirements that discriminate between products. Product categories such as

"compatible" and "not compatible" indicate product compliance to these requirements. Card sorts are a very efficient technique in this situation because only requirements that discriminate between at least two products are acquired. Examples of discriminating customer requirements which might be acquired using card sorts include *"the system shall have a UK-English spell checker"*, *"the system shall allow the user to set a tailored vacation message"*, and *"the system shall allow the user to predefine the font and character size for displayed messages"*. The process chunk for this is:

```
Process-Chunk:
        Name: 1.4
        Goal: Acquire atomic-functional-requirements
        Processes: none
        Situation: no-discriminating(requirements)
        Input-information: content(sub-model(requirements))
        Techniques: {card-sorts}
End-Process-Chunk
```

The contents of the process triplet for process chunk are:

{Acquire-atomic-functional-requirements,no-discriminating-requirements, card-sorts}

Another example of a more complex process chunk defines the *situation "insufficient(behaviour-requirements)"*, which exists when the requirements sub-model contains an insufficient number of behavioural requirements to enable product selection. This process chunk recommends the acquisition of more behavioural requirements using two techniques. The first technique is *use case analysis* for tasks that achieve essential functional requirements (Maiden et al. 1998). The second technique is to use available product demonstration copies as ready-made system prototypes. Software suppliers often offer free simple demonstration copies of their products. The simple product demo copies provide partial working prototypes with which side-by-side product comparisons and facilitated requirements acquisition (e.g. Sutcliffe 1997) can be performed:

```
Process-Chunk:
        Name: 1.5
        Goal: Acquire atomic-functional-requirements
        Processes: none
        Situation: insufficient(behaviour-requirements)
        Input-information: content(sub-model(requirements))
        Techniques:{analyse-use-cases, walkthrough-product-demonstration-
        copies}
End-Process-Chunk
```

Appendix 4f gives the full set of process chunks defined in PORE. The intention is to be able to add more chunks as the method evolves. Although process chunks combine all the method components to achieve each process goal, the iterative nature of the

PORE process makes detecting these chunks very complex. Therefore approaches for guiding the iterative process are needed.

4.8 Summary and chapter conclusions

The theory describes the iterative process guidance using goals, models and situations. The process guidance suggested aim to solve the problems identified in chapter 3. Templates provide the requirements engineering team with process guidance about what to do at three key stages of the process. Templates provide more course grain guidance unlike other process guidance techniques. Goals provide the requirements engineering team with guidance to reject candidate products that are non-compliant with essential customer requirements. Three such high-level goals are defined and five generic processes for achieving each goal are also defined. Products are selected or rejected according to their compliant with the customer's requirements. To enable effective product-requirement compliance mapping, three models are defined. Process chunks and situation rules enable to infer process guidance and situation and the techniques to be used.

The process guidance approaches for guiding the COTS-based development process suggested in this chapter are novel. In particular, they address the lack of process and method guidance for requirements acquisition and COTS software product selection processes which must take place before system design. However, one of the problems is that the iterative process of requirements acquisition and product evaluation/selection is very complex. At any point in this complex process, a large number of possible situations can arise. For example, stakeholders may define a large number of requirements, so the requirements sub-model can give rise to a large number of different situations. The requirements engineering team can also evaluate a large number of software products, so the product and compliance sub-models as well, can give rise to a very large number of situations. In addition, PORE can sometimes recommend a large number of processes and techniques to use in a single situation. To handle this scale of complexity, the theory suggests that a software tool is needed that has a computational model to detect situations and generate process guidance. A prototype concept demonstrator known as PORE Process Advisor is

therefore developed to demonstrate tool support for the PORE approach. The main components of the tool are a process engine which analyses the current set of goals to be achieved (stored in the goal agenda), model properties (inferred by the situation inference engine) and instructions from the requirements engineering team to recommend process advise in the form of the process triplet, i.e. {process-goal, situation, technique-to-use}. The PORE Process Advisor tool is developed with MS Access and Visual Basic. It is designed to integrate with existing software tools such as Rational's RequisitePro requirements management tool (for requirements management) and CREWS-SAVRE tool (e.g. Maiden et al. 1998) for generating scenarios.

Although the method described above addresses hypothesis H2, the effectiveness of the method can not be tested at this point. This is done in chapter 6. Therefore, at this point, hypothesis H2 cannot be fully accepted or rejected.

The following chapter, chapter 5, presents and describes the concept demonstrator tool, PORE Process Advisor.

Chapter 5

The PORE Process Advisor prototype tool Design and Development

This chapter outlines the design and implementation of the process advisor prototype tool that aid in the process of providing the requirements engineering team with process guidance and advice

Chapter 5

Process Advisor prototype tool Design and Development

This chapter outlines the design and implementation of the PORE Process Advisor prototype tool. The tool is a concept demonstrator that exists to demonstrate the feasibility of tool support for the PORE method to provide requirements engineering teams with process advice and guidance according PORE. As described in chapter 4, at any time during the process, a large number of situations can be generated and a large number of process chunks and techniques can be recommended. Some process situations are complex. A software tool is needed to detect them. To handle the scale and complexity of these situations the software tool has a computational model that infers situations to generate process guidance. Section 5.1 describes the tool's architecture and its main components, functionality and interface. Section 5.2 describes its implementation and examples. Section 5.3 presents a 'scenario walkthrough' to demonstrate how the components of the tool are linked and how process advice and guidance is computed.

5.1 Architecture

The PORE concept demonstrator prototype tool was built with MS Visual Basic and Access under Windows 95. The prototype tool is designed to be linked or integrated with other tools including Rational's RequisitePro requirements management tool and City University's CREWS-SAVRE scenario walkthrough prototype (Maiden et al. 1998). The tool's components are shown in Figure 5.1:

- a process engine that determines the goals to be processed (discussed in section 4.2 & 4.3);
- a database that stores: (a) situations inferred from the situation model (discussed in section 4.4 & 4.5); (b) process chunks which prescribe process techniques; (c) a method box that provides process guidance;
- a process advisor that provides guidance and advice provided in the method box (discussed in section 4.6);

- an inference engine that uses process situation rules to infer situations that trigger process guidance (discussed in section 4.7 & 4.8).

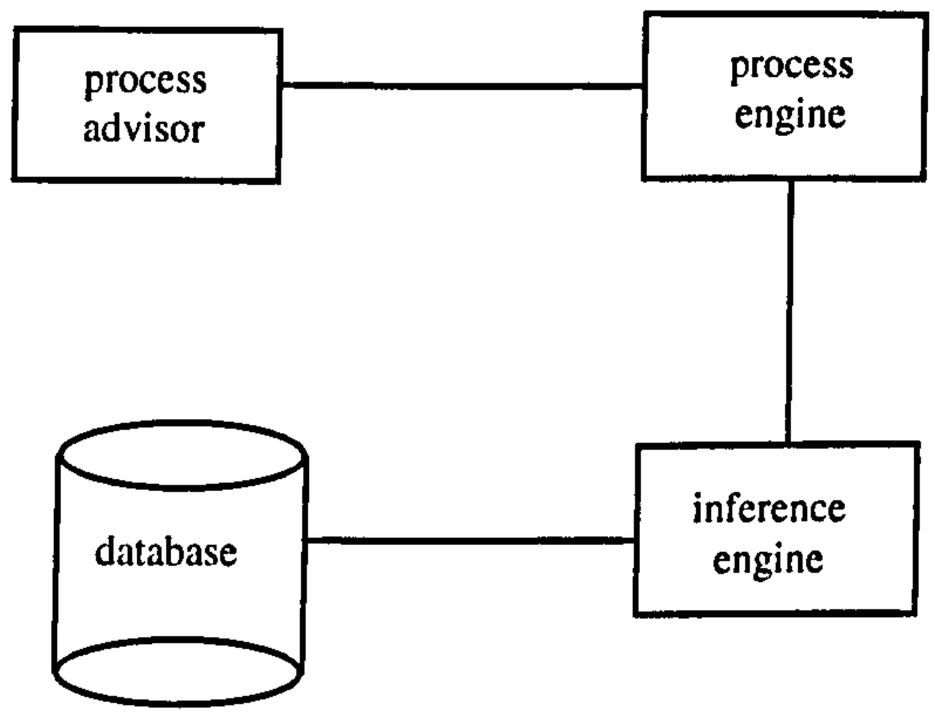


Figure 5.1: Architecture of PORE process advisor prototype tool. The figure shows static associations between the prototype's 4 main components that are linked to provide process guidance and advice to the evaluation team.

The following sections provide a detailed description of the prototype's 4 main components.

5.1.1 Process Engine

The process engine algorithm links: (a) situation rules that infer process situations that trigger process guidance and advice; (b) situation model from which the rules infer situations; and (c) process chunks that prescribe process techniques and presents process guidance and advice. The process engine it analyses the current set of goals to be achieved, the model properties that are inferred by the situation inference engine and the instructions from the requirements engineering team to recommend process advise in the form of the process triplet, (<*process-goal, situation, technique*>) modelled as process chunks. Its main function is to check whether processes have been completed successfully by checking that the situations that triggered the process do not currently exist. If the situations no longer exist, the algorithm logically identifies new processes to be undertaken to achieve the next process goal. It uses the current goal to be achieved and the current situation to recommend suitable techniques from the method box, then presents this triplet to the process advisor.

Whenever the evaluation team requests advice, the process engine algorithm does three basic computations:

- it infers whether the processes in the current-process lists have been completed successfully. It does this by attempting to infer whether the situation or the many situations that triggered these processes still exist;
- if these situations no longer exist, it identifies new processes to undertake to achieve the next ordered goal;
- it uses the lists to compile process advice from the method box and present it to the team.

A more precise specification of the process engine algorithm is shown in Figure 5.2 below.

```
DO accept-advice-request
• COPY new-goalprocess-list TO old-goalprocess-list
• COPY new-contextprocess-list TO old-contextprocess-list
• SET new-goalprocess-list=[]
• SET new-contextprocess-list=[]

DO check-current-advice (**to check whether previous processes complete**)
• REPEAT FOREACH process in new-contextprocess-list
• • RETRIEVE process-chunk WHERE goal=member[old-contextprocess-list]
• • IF situation of process-chunk=true
• • • ADD process to new-contextprocess-list
• • ENDIF
• ENDREPEAT
ENDDO
DO check-new-contextprocess-list
• IF new-contextprocess-list•[]
• • GO TO PREPARE-ADVICE
• ENDIF
ENDDO

DO determine-new-advice (**to determine new processes to complete**)
• WRITE next goal-driven process to new-goalprocess-list
• REPEAT FOREACH process in next-goaldriven-process
• • RETRIEVE process-chunk WHERE goal=process(next-goaldriven-process)
• • IF situation of process-chunk=true
• • • ADD process to new-contextprocess-list
• • ENDIF
• ENDREPEAT
ENDDO

DO PREPARE-ADVICE (**to determine technique/process guidelines**)
• REPEAT FOREACH process in new-contextprocess-list
• • RETRIEVE process-chunk
• • WRITE technique to new-technique-list
• ENDREPEAT
• DO show-advice
ENDDO
```

Figure 5.2: PORE's process engine algorithm

5.1.2 The Process Advisor

The process advisor uses situation rules to produce process guidance and to determine what advice, techniques, tools and methods to recommend to the requirements engineering team. It also validates the advice to be recommended based on the current state of the situation model. The process advisor analyses the current set of goals to be achieved, properties of the situation model that are inferred by the situation inference engine, and instructions from the requirements engineer to provide process guidance and recommend advice about what to do next. Guidance and advice is given using the <situation, advice, technique> triplet. The job of the process advisor is to handle the reasoning portion of the prototype tool. Instead of the tool providing the team with a list of all current situations, the process advisor uses the situations stored in the database to guide them based on the specific situations and defined process logic. The process advisor iteratively provides the team with guidance and tailors the advice based on the information provided. Figure 5.3 below depicts how the user interacts with the process advisor

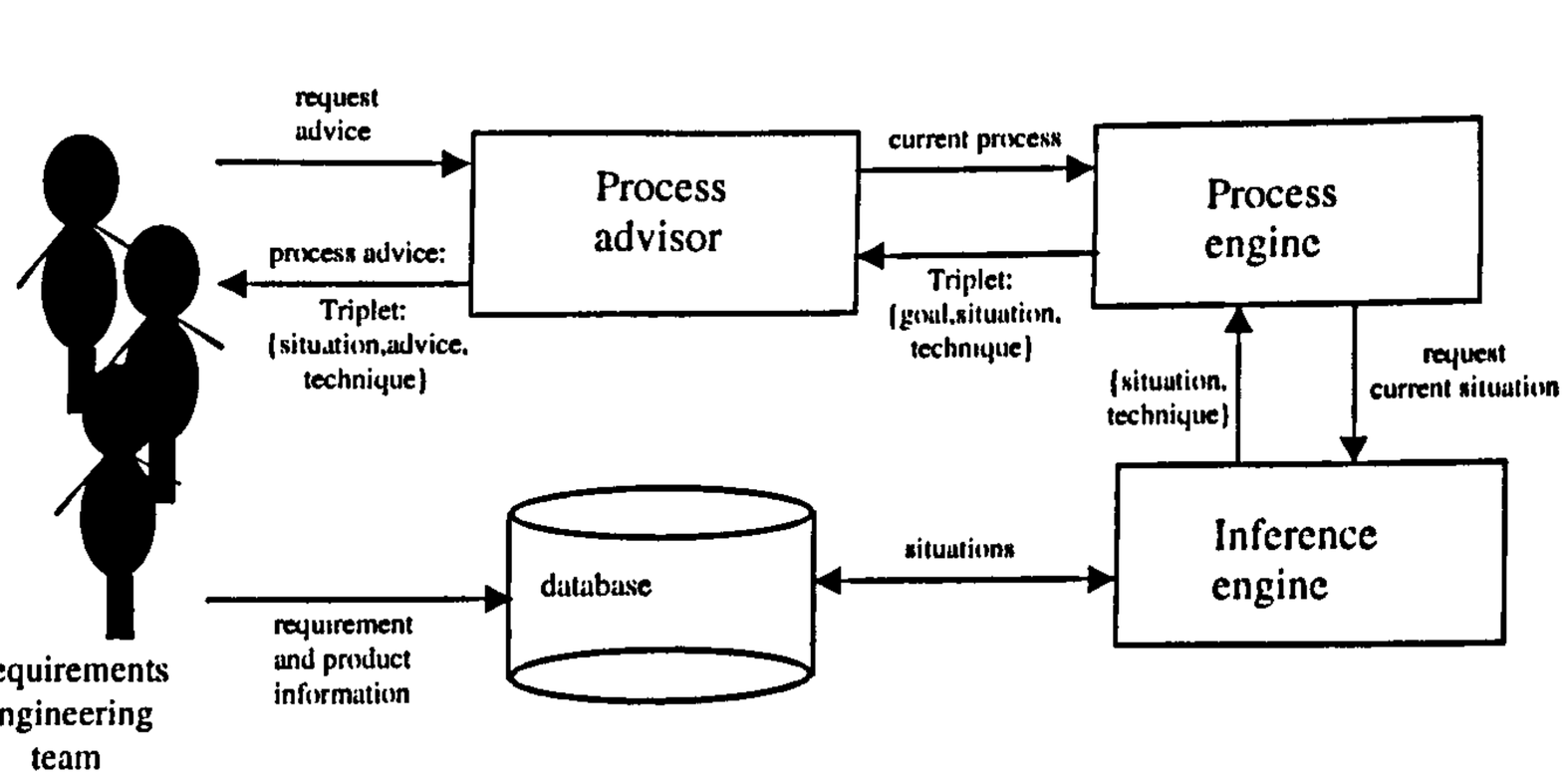


Figure 5.3. The involvement of the user in the process advisor tool. The figure shows the information flows between the 4 main components of the prototype. When the user requests advice the process advisor determines current process. The process engine then uses the detected current process to determine current situation using the inference engine. The inference engine infers the current state of the compliance model to determine the current situation. The process engine then passes the triplet <goal, situation, technique> to the process advisor. The process advisor uses the triplet to produce advice which is presented to the user.

5.1.3 The Inference Engine

The inference engine is the most important component of the prototype tool. It acts as the control unit and does all the reasoning about the current state of the models and what advice to offer next. The reasoning consists of matching the current situations to the process goal and suitable techniques to handle the situation. Advice is then given in the form of a triplet and processed as process chunks. The inference engine uses several rulesets to guide the process logic reasoning. Rules are fired in two ways:

- in response to a specific situation, which in turn depends on the data entered by the team or generated by a rule that has been executed in the inference engine, or
- executed to attain a specific goal, i.e. specific goal directs the process, giving it a focused and tightly controlled execution strategy.

The situation rules are represented in the form of a logical implication, ***IF <condition> THEN <action>***. The rule conditions assert process goals that match the conditions. The goals asserted persist as long as there are situations that remain relevant to the goal. The inference engine identifies rules that apply to the current goal based on the inferred situation. The rules then trigger process advice that guides the team to add new information into the databases, although the team might not necessarily follow the advice.

5.1.4 Database

The database is the operationalisation of the PORE's situation model described in section 4.5 of chapter 4. The database stores the inferred situations and includes the method box which holds information about available techniques, methods and tools. The database is developed using MS Access. MS Access was chosen because it is a COTS product that is available in the market and integrates easily with Visual Basic, the development environment for the prototype. The database consists of a number of individual tables. Tables 1 – 4 show examples of the requirements table, feature table, advice table and techniques table respectively.

Table 4.1 shows the requirements database table that implement part of the requirement model shown in Figure 4.6 in chapter 4.

RequirementID	Type	Description	Priority	Source	Fit-criteria
R1	Functional	The system shall send a	High	LB	The systems s
R10	NonFunctional	The system shall have a	High	LB	The system
R100	Functional	The system shall allow the	Low	LB	The system
R101	Functional	The system shall enable to	Medium	SG	The system
R102	Functional	The system shall enable	Medium	MN	The system
R103	Functional	The system shall enable	Medium	LB	The system
R104	Functional	The system shall be able	High	MN	The systems s
R105	Usability	The system shall enable	Low	SG	The system
R106	Functional	The system shall allow the	High	SG	The system
R107	Functional	The system shall be able	Low	MN	The system
R108	Functional	The system shall have a	High	LB	The system
R109	Functional	The system shall enable to	Medium	MN	The system
R11	NonFunctional	The system shall hide	Medium	LB	The systems s
R110	Functional	The system shall allow to	High	LB	The system
R111	Functional	The system shall allow to	High	MN	The system
R112	Functional	The system shall allow to	Low	SG	The system
R113	NonFunctional	The system shall have	Medium	SG	The system
R114	Functional	The system shall enable	High	MN	The system
R115	Functional	The system shall be able	High	MN	The system
R116	Functional	The system shall have a	Low	SG	The system
R117	Functional	The system shall enable	High	LB	The system
R118	NonFunctional	The system shall store all	Low	SG	The system

Table 5.1: Example of the requirements database as represented in the PORE database. Each requirement has a unique identifier and type as shown in figure 4.6.

The following table shows a sample of the product features as defined in the product meta-model. The table shows that product features are specialised into components, functions or actions, indicated by the feature type.

FeatureName	ProductName	FeatureType
AddressBook	OutLook	Component
Addresses	OutLook	Function
Apointments	Pegasus	Function
AutoReminder	OutLookExpres	Function
AutoReminder	Pegasus	Function
AutoReminder	Eudora	Function
AutoSpellChecker	OutLookExpres	Function
AutoVacationRepl	OutLookExpres	Function
AutoVacationRepl	Pine	Function
AutoVacationRepl	OutLook	Function
AutoVacationRepl	Eudora	Function
Calender	OutLookExpres	Component
Calender	Communicator	Component
Calender	Pegasus	Component
Calender	Eudora	Component
Diary	OutLookExpres	Component
Diary	Pegasus	Component
Dairy	Eudora	Component
Dictionary	OutLookExpres	Component
Dictionary	Eudora	Component
Dictionary	Pegasus	Component
Dictionary	OutLook	Component

Table 5.2: Example of the product feature table. The features are mapped to requirements in the requirements sub-model to determine product-requirement compliance. The table also shows which product possesses the feature and their type.

The following table shows a sample of the *'process situations'* and corresponding advice for guiding the requirements engineering about what to do next. The 'situations' are key to the process guidance and advice that is presented to the requirements engineering team and for selecting suitable techniques, methods or tools from the method box. The situations are part of PORE process chunks and part of the multi-layered process guidance that is given to the requirements engineering team in the form of triplet.

SituationName	Advise
EmptyRequirements	Acquire customer requirements
EmptyProducts	Identify candidate products
InsufficientRequirements	Acquire more customer requirements
InsufficientProductFeatures	Do more detailed product analysis
NonDescriminatingRequirements	Acquire detailed discriminating requirements
EmptyFeatures	Acquire product features
InsufficientSupplierRequirements	Acquire more supplier requirements
InsufficientContractualRequirements	Acquire more contractual requirements
InsufficientArchitectureRequirements	Identify and acquire more architecture requirements
InsufficientUsabilityRequirements	Acquire usability requirements
NoComplianceMapping	Determine compliance mapping
DecisionMaking	Determine product ranking
NonDescriminatingProductFeatures	Acquire more product features

Table 5.3: Example of the PORE situations. The situations and advice are inferred by the rules in the process chunks.

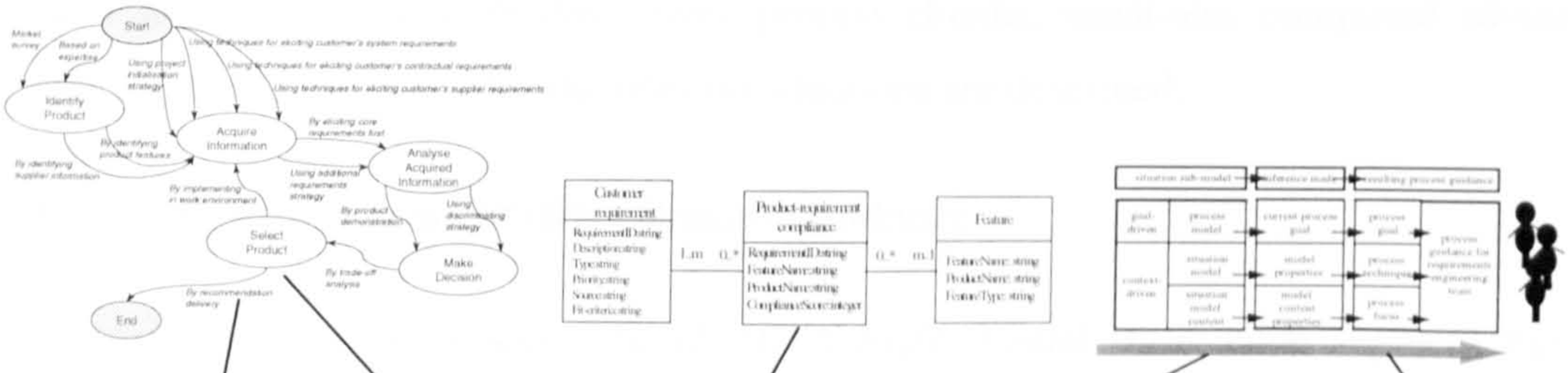
For each situation, PORE identifies suitable advice. The process engine and the inference engine infer current situation using situation rules and the current state of the situation model and the process advisor recommends suitable advice to the team about what to do next. New situations and advice are added, as they become relevant.

The following table shows how the PORE method box is implemented. The table shows information about techniques and methods that are provided in the PORE method and the process situations that the techniques aim to solve. Existing COTS-based development methods discussed in section 2.9 and decision-making techniques discussed in section 2.10 are also included in the PORE method box. The table also shows links between the techniques or methods in the method box and the process situations identified in Table 5.3. This link is indicated by the common attribute 'SituationName' that is defined in both tables.

TechMethodName	SituationName	TechStrength/Weakness
Brain Storming	No requirements	Brainstorming.htm, ACRE
Scenario Analysis	Insufficient requirements	Scenario.htm, ACRE
OTSO	No candidate products	OTSO.htm
CISD	Evaluate COTS products	CISD.htm
Feature Analysis	Identify more requirements	FA.htm
IIDA	Evaluate products	IIDA.htm
IusWare	Decision analysis	IusWare.htm
Interviews	Insufficient requirements	Interviews.htm, ACRE
Card Sorting	Non discriminating requirements	Cardsort.htm, ACRE
Prototype Analysis	Insufficient requirements or product	Prototype.htm, ACRE
Product Demo	Insufficient product information	Product-demo.htm
Architecture Analysis	Insufficient architecture	Arch-Analysis.htm
Use Case Walkthrough	Insufficient requirements	Use-case.htm, ACRE
Questionnaire	Insufficient or no product information	Qurstionnaire.htm
Internet	No products, All products eliminated	Internet.htm
Market Survey	Insufficient or no products	Market-survey.htm
SAAM	Insufficient architecture	SAAM.htm
ATAM	Insufficient architecture	ATAM.htm
RAD	Insufficient requirements	RAD.htm, ACRE
Compliance Mapping	No compliance mapping	Compliance.htm
AHP	Decision-Making	AHP.htm
MCDA	Decision-Making	MCDA.htm
Out Ranking	Decision-Making	OutRanking.htm
Weighted Score Method	Decision-Making	WSM.htm

Table 5.4: Sample of the information about techniques and methods that are provided in the PORE method box.

The method box includes currently available techniques and methods that can address process situations. For each technique, the method box identifies the situation(s) in which the technique is suitable. The method box is designed so that new methods and techniques can be added, as they become available in the market. The method box is designed to be linked to the publicly available ACRE framework (Maiden & Rugg 1996). Figure 5.4 below shows the link between some of the tables that operationalises the PORE models and the theory components described in chapter 4.



Process Goal	Situation	Advice	Technique
Reject products non-compliant with goal 1	None	None	None
Acquire atomic functional requirements	Empty (requirements sub-model)	Acquire atomic customer functional requirements	Brain storming, use cases, interviews, scenarios
Acquire atomic functional requirements	Insufficient functional requirements	Acquire more functional requirements	Use-case-walkthrough, structured-interview, scenario-analysis,
Acquire product information	Empty (product sub-model)	Identify candidate products and acquire product and supplier information	Questionnaire, internet, market survey
Acquire product information	Insufficient (product information)	Do more details analysis of candidate products	Walk-through-product-demonstration-copies, request-for-information
Analyse acquired requirements and product information	None	None	None
Analyse product-requirement compliance	Sufficient (requirements and product information)	Analyse acquired product information and requirements for completeness	Scenario-analysis, fit criteria, product prototype
Determine requirement-product compliance	No(requirement-product compliance)	Determine-compliance mapping	Compliance-checklist, compliance-walkthrough
Analyse product-requirement compliance	Non-discriminating requirements	Acquire more discriminating customer requirements	Card sorts
Select/Reject one or more non-compliant products	Small decision space	Make decisions about product-requirement compliance	Out-Ranking, MCDA, AHP
Redo requirement/product analysis or identify new candidate products	All products eliminated	Do detailed analysis of the candidate products and or identify new products or relax customer requirements	Product-prototypes, internet, data-analysis, market-survey

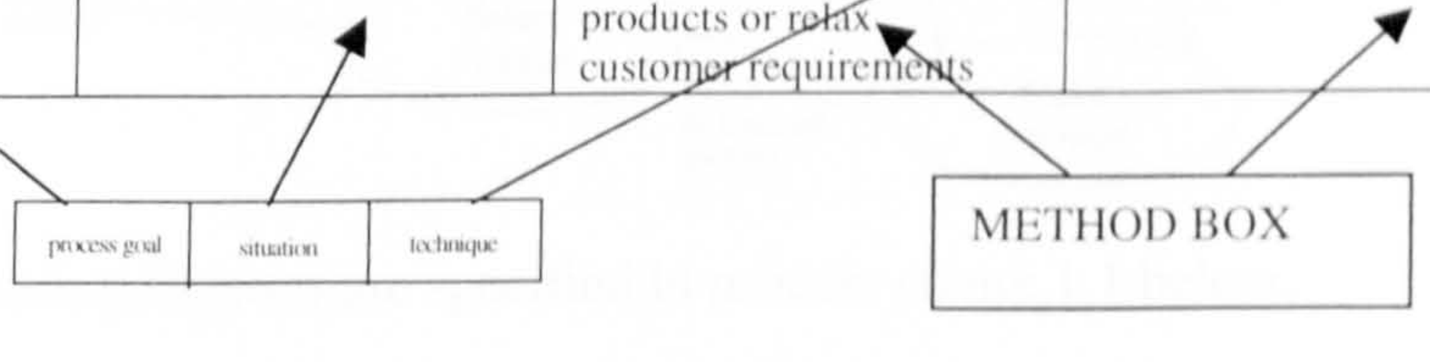


Figure 5.4 The relationships and links between the method components and the tool implementation.

The case study from chapter 4 is used to demonstrate PORE's process guidance and the underlying computational mechanisms. It reports acquisition of requirements to select a COTS electronic-mail system for a university office. The following examples

show snapshots of processes that were needed to make key selection decisions. For each snapshot, relevant process goals, process chunks, situations, computed advice and situation rules that infer the relevant situations are described.

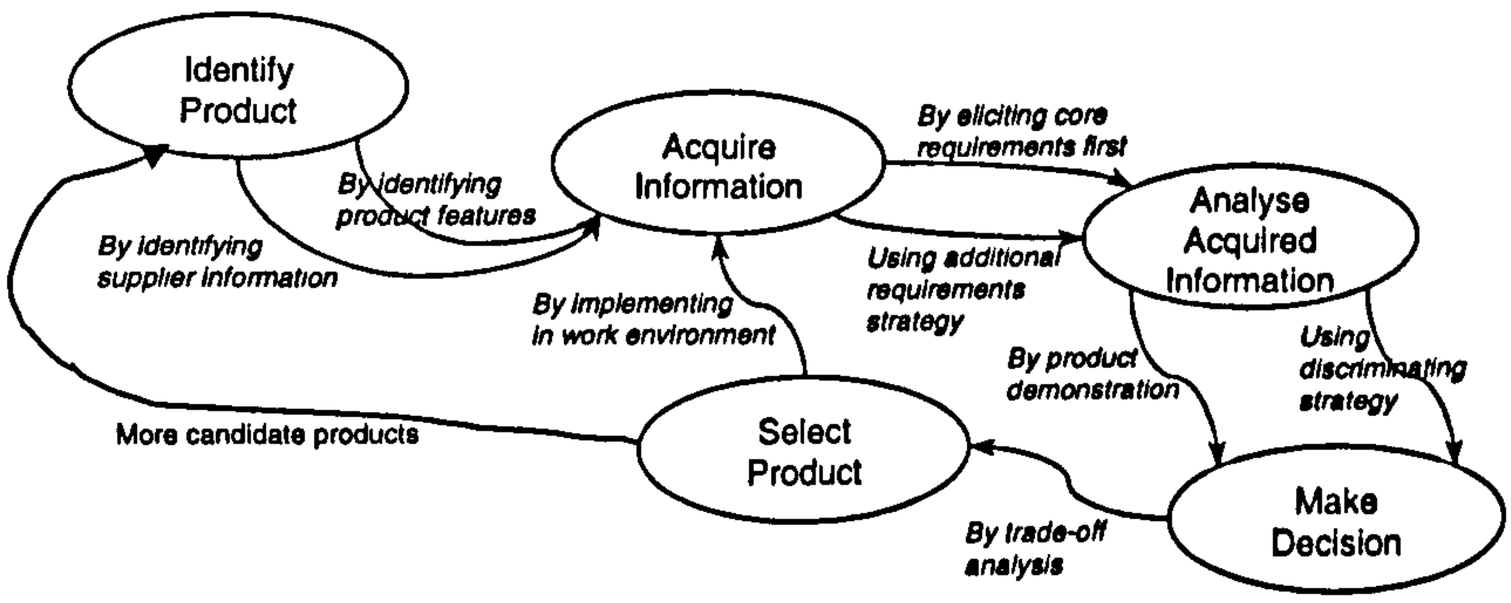
5.2 Demonstrating PORE's Process Guidance

At each stage of the process, the *IF-THEN-ELSE* Visual Basic code segment that implements the rules that infer the current state of the situation model (i.e. situation) and that trigger the relevant process chunk is shown.

As described in chapter 4, the first key decision to make in the PORE process is to reject candidate products according to goal-1:

- *to reject candidate COTS products that are non-compliant with, atomic customer requirements*.

To achieve this goal, all 5 PORE goal-driven generic processes have to be completed for the goal. At any stage, the relevant generic process that is currently being undertaken is indicated by shading that particular process. The diagram below shows that no process is currently being undertaken.



The goal-driven generic processes are specified in process chunk 1.1 below.

```
Process-Chunk:
      Name: 1.1
      Goal: Reject products non-compliant with goal-1
      Processes: Identify products AND Acquire information from
            stakeholders THEN
            Analyse acquired information THEN
            Determine non-compliant products THEN
            Reject one or more non-compliant products
      Situation: None
      Input-information: None
      Technique: None
End-Process-Chunk
```

To complete each cycle (i.e. to reject products non-compliant with customer's atomic requirements), the four processes of chunk 1.1 are undertaken in a strict sequence as indicated by the 'THEN' statements in the chunk. As this chunk describes the high-level generic process, there are no triggering situations, information input and recommended techniques. Each process in turn has a unique process chunk, thus specifying hierarchies of context-driven processes.

At this stage of the process, when the requirements engineering team asks for advice, the process engine analyses the instructions from the prototype's user and the current sets of goals to be achieved. It then uses the inference engine to analyse the situation model properties. The inference engine in turn uses the situation rules to reason about the current state of the situation model to infer current situations. It then matches the existing situations to the current process goal and techniques and presents them to the process engine. The process engine then uses the rules to trigger the relevant chunks and presents the chunk to the process advisor in the form of <*process-goal, situation, techniques*> triplet. The process advisor analyses the contents of the triplet and uses the *'situation'* part of the triplet to recommend guidance and advice to the team about what to do next. This advice is presented to the requirements engineering team in the form of <*situation, advice, technique*> triplet. At this stage the team might choose to follow the advice as presented or to ignore it.

At the beginning of the process to select an e-mail software package the RE team asks for advice and guidance from the process advisor tool. The process engine examines the situation model and deduces that the requirement sub-model, product sub-model and the compliance sub-model are all empty. The inference engine then infers the situations: *empty(requirements sub-model) and empty(product sub-model)* using the following situation rules:

```
If RequireVal = 0 Then
    Call produceAdvice("EmptyRequirements", " Empty requirement
    model")


If ProductVal = 0 Then
    Call produceAdvice("EmptyProducts", "Empty product model")
```

It recommends the use of the techniques *{interview, use-case analysis, brainstorming}* for requirements acquisition and *{internet, market survey, questionnaire}* for identifying candidate products. The process engine then triggers process chunks 1.2 and 1.3 to be processed and present them to the process advisor as triplets.

```
Process-Chunk:
      Name: 1.2
      Goal: Acquire atomic-functional-requirements
      Processes: none
      Situation: empty(requirements sub-model)
      Input-information: none
      Techniques: {interview, use-case analysis, brainstorm}
End-Process-Chunk


Process-Chunk:
      Name: 1.3
      Goal: Acquire product-information
      Processes: none
      Situation: empty(product sub-model)
      Input-information: none
      Techniques: {questionnaire, internet, market-survey}
End-Process-Chunk
```

The process advisor uses the advice triplet 'situations' part to compute the advice that is then presented to team to ' *identify candidate products*' and to '*acquire atomic functional requirements*'. The advice is presented to the RE team in the form <situation, advice, techniques> as shown in Figure 5.5.

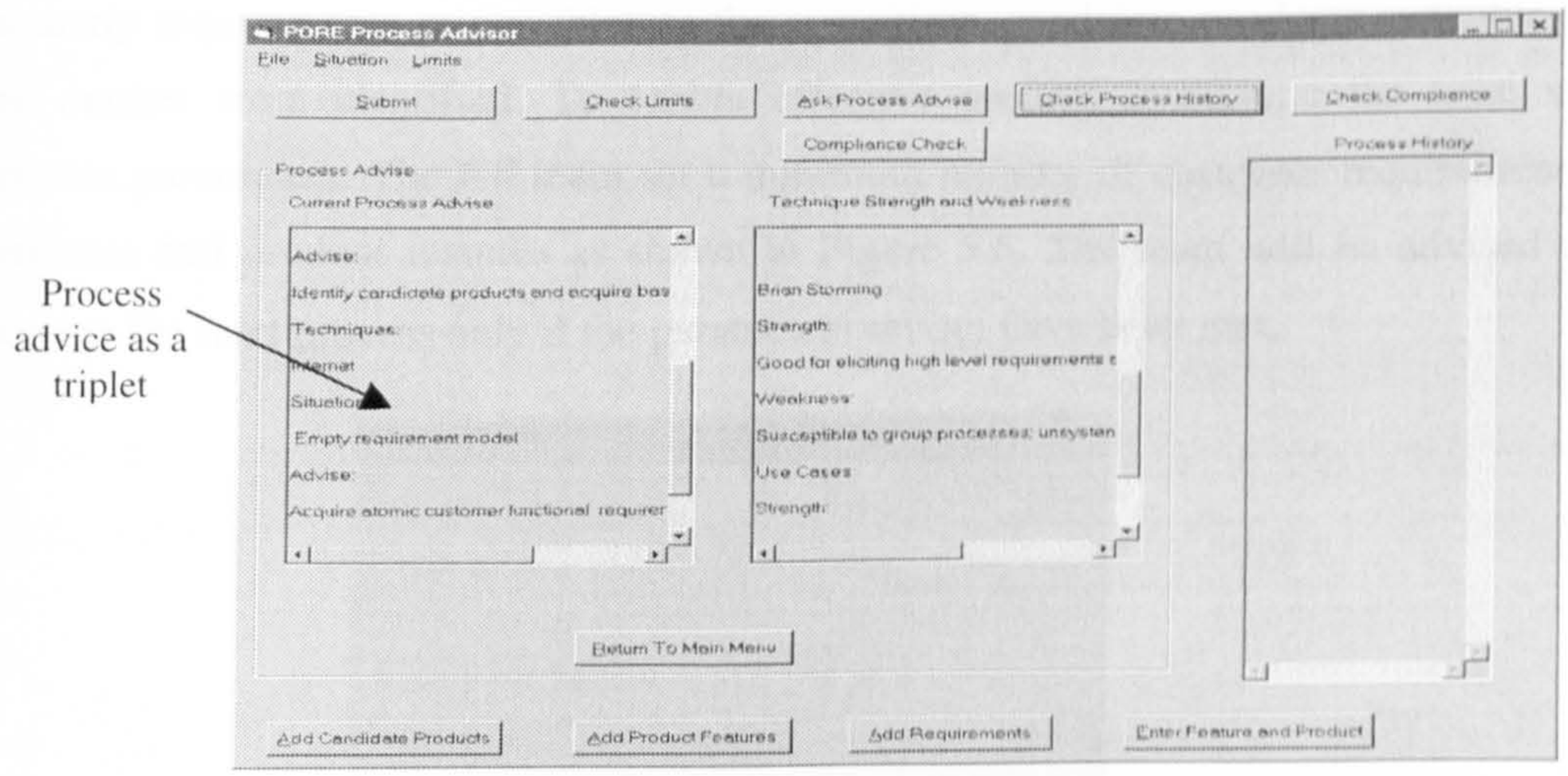
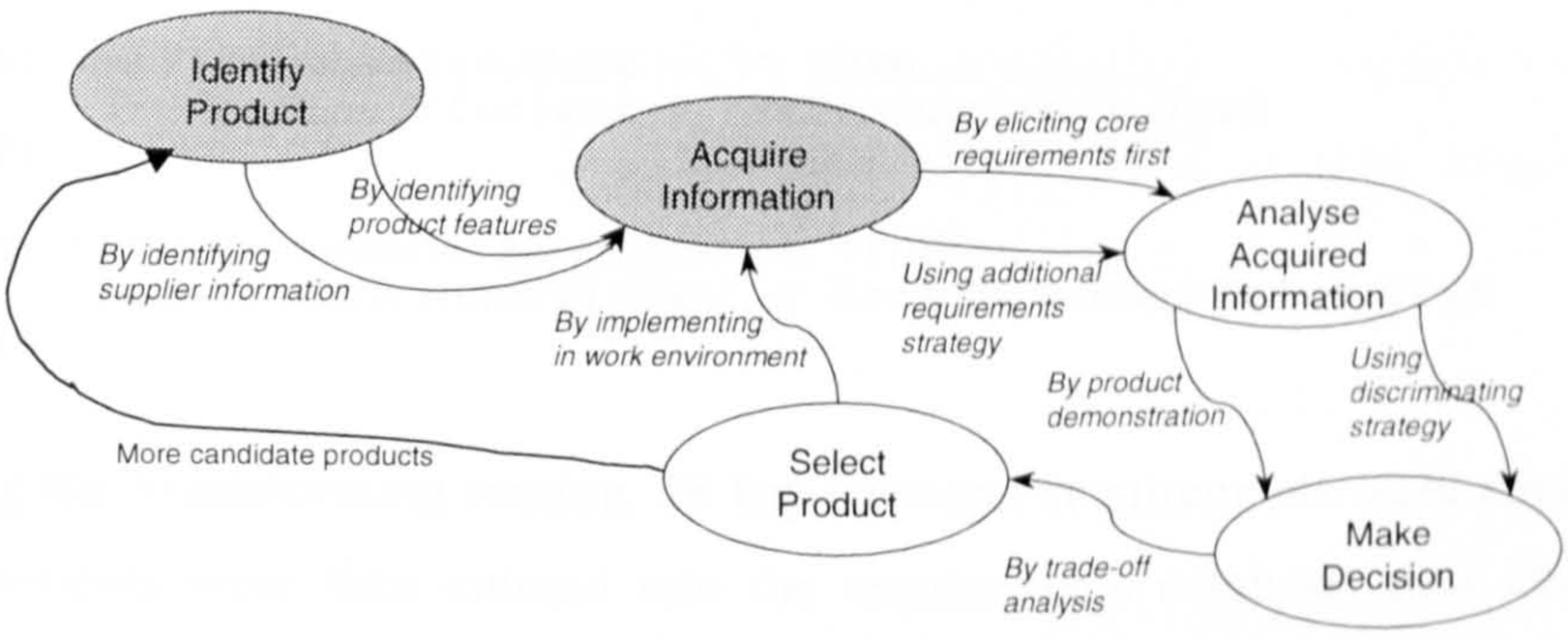


Figure 5.5: Process advice provided the prototype tool advising the evaluation team to identify more candidate COTS products and to acquire atomic customer requirements. The advice includes suitable techniques for solving the situation and description of each technique's strength and weaknesses.

The RE team followed the process advice and guidance and executed the first 2 of the five generic processes, i.e. *identify products and acquire information from stakeholders*, as indicated below:



After this, an initial brainstorming session with three stakeholders was held to acquire first-cut requirements for the office e-mail system. The stakeholders were experienced users of electronic mail and the internet. A facilitator chaired the session and elicited as many requirements as possible. At the same time, candidate products available in the market were identified. To enable effective product evaluation the team set process parameters. The RE team set a minimum number of customer requirements, products and product features as shown in Figure 5.6. The team will be advised to execute the next process only if the parameters set out have been met.

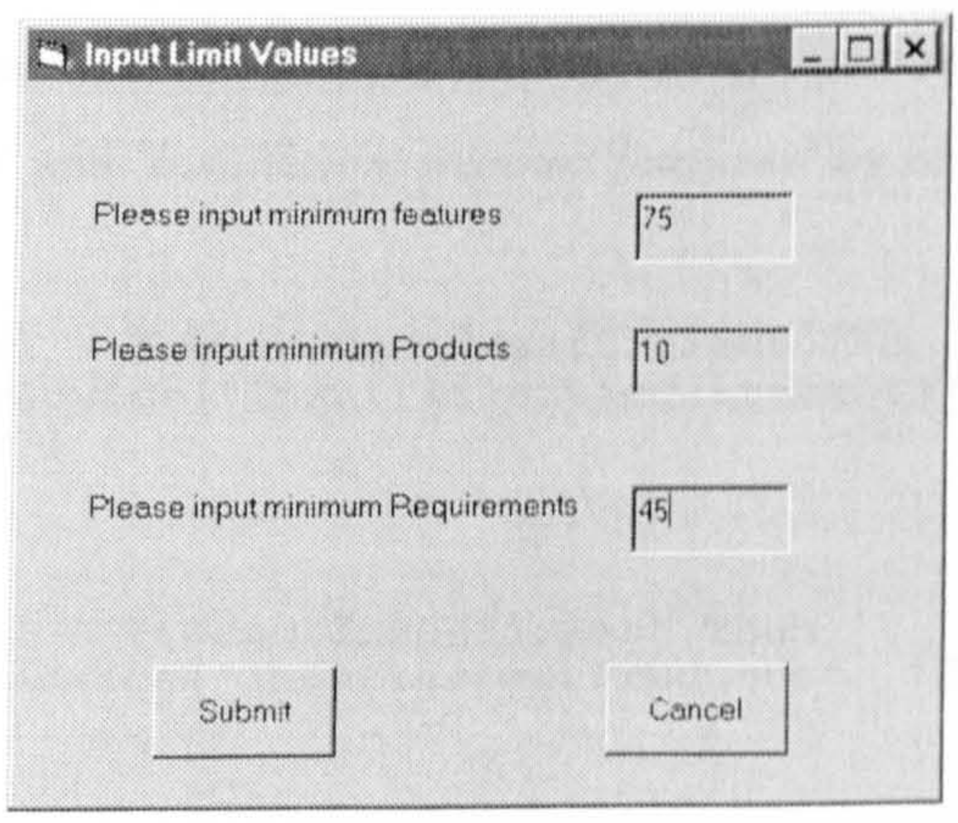


Figure 5.6: Setting minimum values for candidate products and customer requirements. This will help the team to decide when to start analysing the acquired information.

The following rules are processed to determine if the number of acquired requirements and product information is greater than the set limits:

```
If Not txtFeatureValue.Text = "" Then
        FeaturesInsufficient = txtFeatureValue.Text
End If

If Not txtProductValue.Text = "" Then
        ProductInsufficient = txtProductValue.Text
End If

If Not txtRequirementValue.Text = "" Then
        RequirementInsufficient = txtRequirementValue.Text
End If
```

During the brainstorming session, 28 key customer requirements were elicited. These requirements were then entered into the requirements database, thus changing the state of the requirement sub-model. A market survey was done using the internet and questionnaire to identify candidate products and acquire product information. Information about products and their features received from suppliers was entered into the database, thus also changing the state of the product sub-model. After entering the information and therefore changing the states of the product and requirement sub-models, the team signaled their completion and asked for process advice and guidance.

The number of acquired customer requirements and the number of candidate products identified and the number of product features was lower than the predefined limits, the inference engine inferred the situations: *'insufficient requirements'*, *'insufficient candidate products'* and *'insufficient product features'* by triggering the three rules:

```
ElseIf RequireVal < RequirementInsufficient Then
    Call produceAdvice("InsufficientRequirements", "Insufficient
    requirements")
End If


ElseIf ProductVal < ProductInsufficient Then
    Call produceAdvice("InsufficientProducts", "Insufficient
    products")
End If
```

```
ElseIf FeatureVal < FeaturesInsufficient Then
    Call produceAdvice("InsufficientProductFeatures", "Insufficient
    product features")
End If
```

The process engine uses the current situations *'insufficient requirements'*, *'insufficient candidate products'* and *'insufficient product features'* to infer that the original process goal *'acquire information from stakeholders'* has not been met. Based on these situations, the inference engine recommends the techniques: *{use-case-analysis, use-case-walkthrough, scenario-analysis, interviews, product-demo-copies}* for requirements acquisition and *{internet, market-analysis, request-for-information, questionnaire}* for identifying more candidate products. It then triggers process chunks 1.4 & 1.5 and presents them to the process advisor:

```
Process-Chunk:
      Name: 1.4
      Goal: Acquire atomic-functional-requirements
      Processes: none
      Situation: insufficient(functional-requirements)
      Input-information: content(sub-model(requirements))
      Techniques:{analyse-use-cases,walkthrough-product-
      demonstration-copies}
End-Process-Chunk
```

```
Process-Chunk:
      Name: 1.5
      Goal: Acquire product information
      Processes: none
      Situation: insufficient(products-information)
      Input-information: content(sub-model(product))
      Techniques:{internet, market-analysis, request-for-information)
End-Process-Chunk
```

The process advisor then used the current situations to advise the team to *'acquire more atomic customer functional requirements'*, and to *'acquire more candidate products information'* using the recommended techniques. The advice provided to the requirements engineering team becomes the new goals for the process chunks. Figure 5.7 shows the advice as presented by the prototype tool to the team.

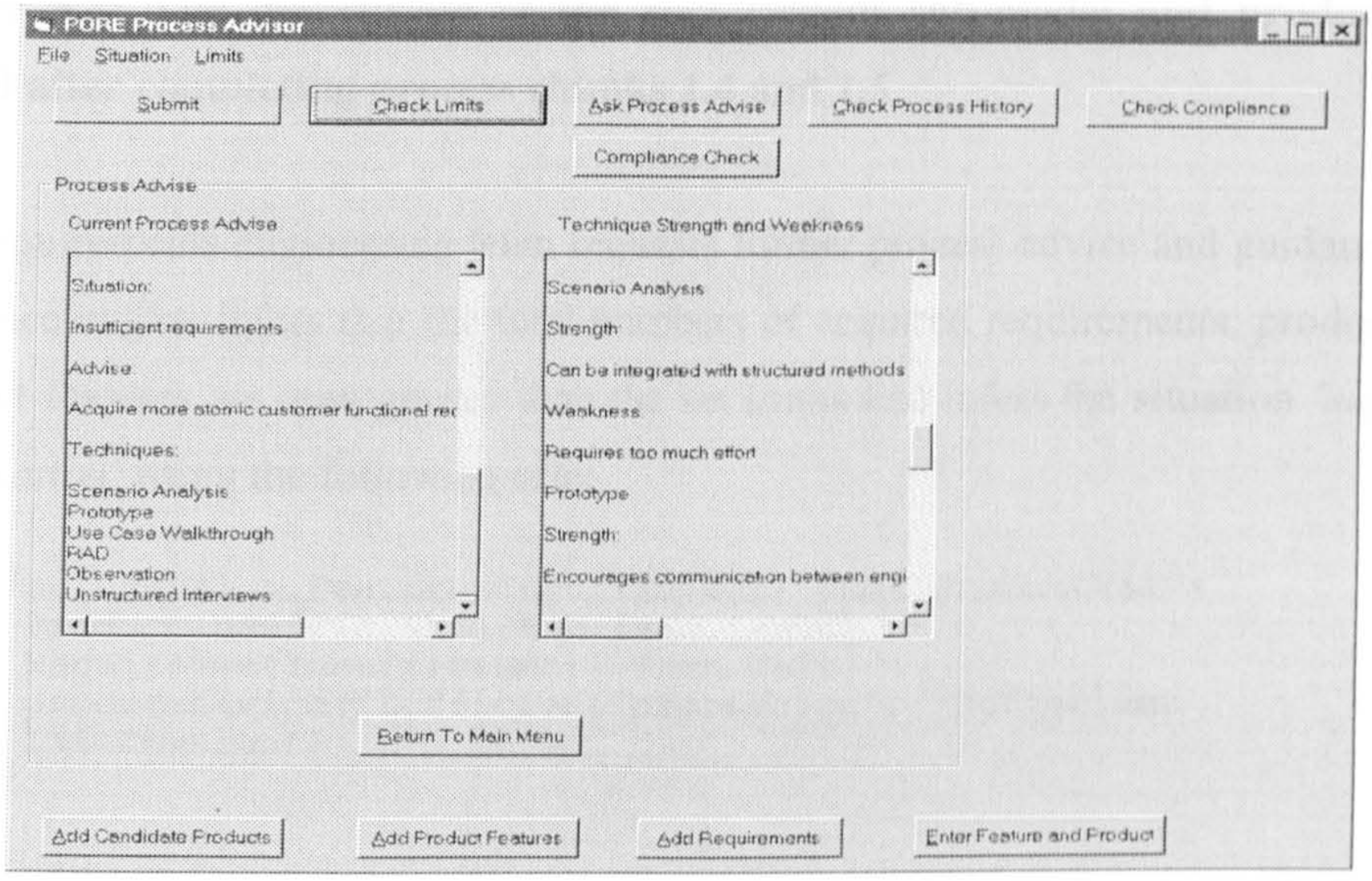


Figure 5.7: Dialogue between the PORE Process Advisor and the requirements engineering team to seek guidance from the process engine. The Process Advisor engine detects that there are insufficient requirements and therefore advises the requirements engineering team to acquire more customer requirements and recommends some suitable techniques.

The team followed the recommended advice and entered more acquired requirements and product information into the database as shown in Figure 5.8. This changes the state of the product and requirements sub-models.

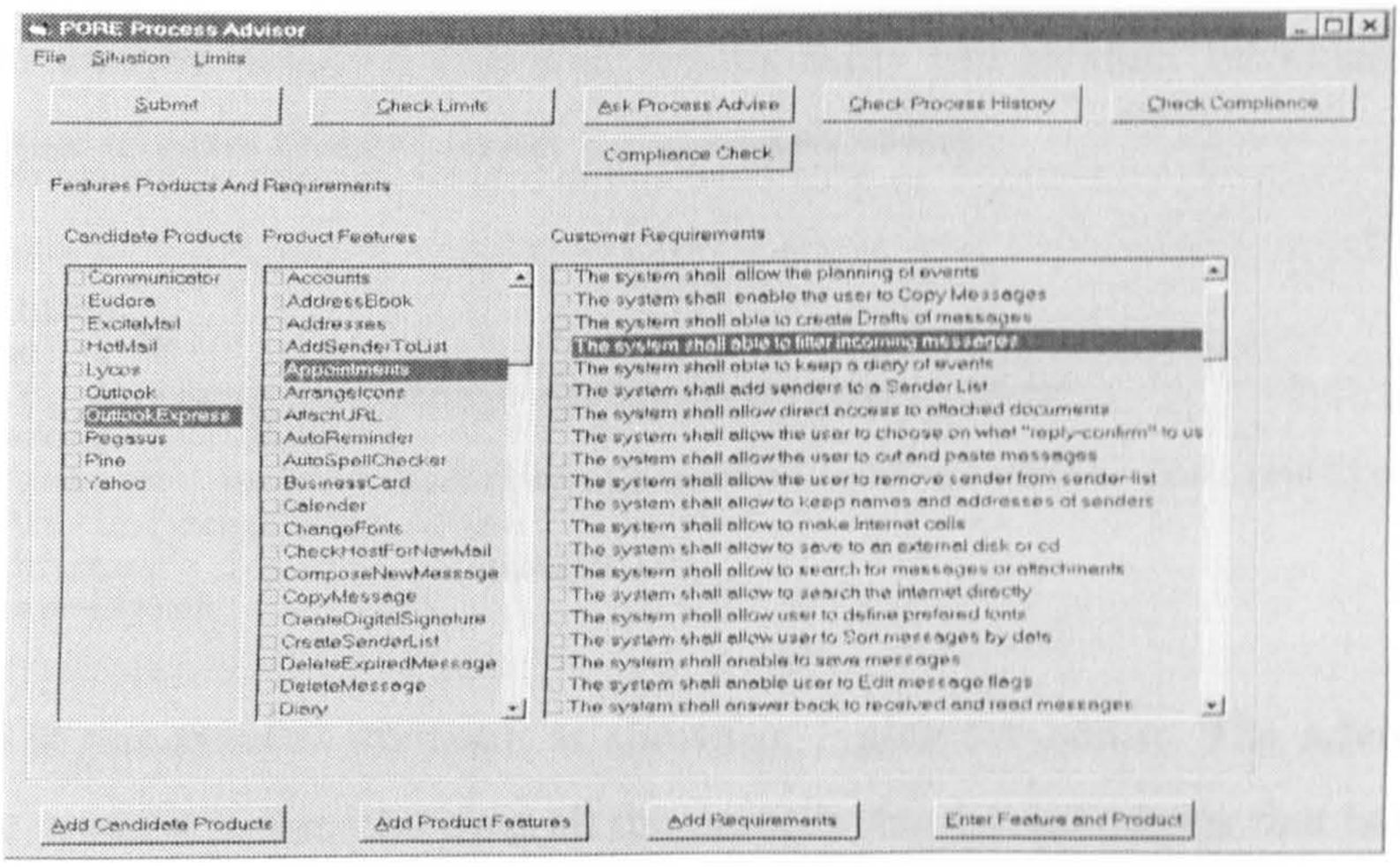


Figure 5.8: The description of the requirement sub-model and product sub-model after completing process chunks 1.4 and 1.5.

The requirements engineering team requests further process advice and guidance. The inference engine infers that the total numbers of acquired requirements, products and product features are now greater than the set limits and infers the situation *'sufficient information'* using the following rule:

```
If ((FeatureVal > FeaturesInsufficient) And (ProductVal >
      ProductInsufficient) And (RequireVal >
      RequirementInsufficient)) Then Call
      produceAdvice("SufficientInformation", "Sufficient
      Information")
End If
```

This indicates that the previous situation no longer exists. The process engine then determines the next goal-driven process to '*analyse acquired information*' which is specified in process chunk 1.6.

```
Process-Chunk:
      Name: 1.6
      Goal: Analyse acquired information
      Processes:  Analyse product-requirement-compliance
      Situation: None
      Input-information: None
      Technique: None
End-Process-Chunk
```

The process engine presents the triplet *<analyse-acquired-information, sufficient information, scenario-analysis>* to the process advisor. The process advisor advises the team to analyse acquired customer requirements and product information. The process engine triggers process chunk 1.7 to be processed:

```
Process-Chunk:
      Name: 1.7
      Goal: Analyse product-requirement-compliance
      Processes:  None
      Situation: sufficient(requirements and product information)
      Input-information: None
      Technique: {scenario analysis}
End-Process-Chunk
```

The advice is presented to the team as shown in Figure 5.9 below. The advisor also presents the process history showing all the process chunks and advice that have been recommended so far.

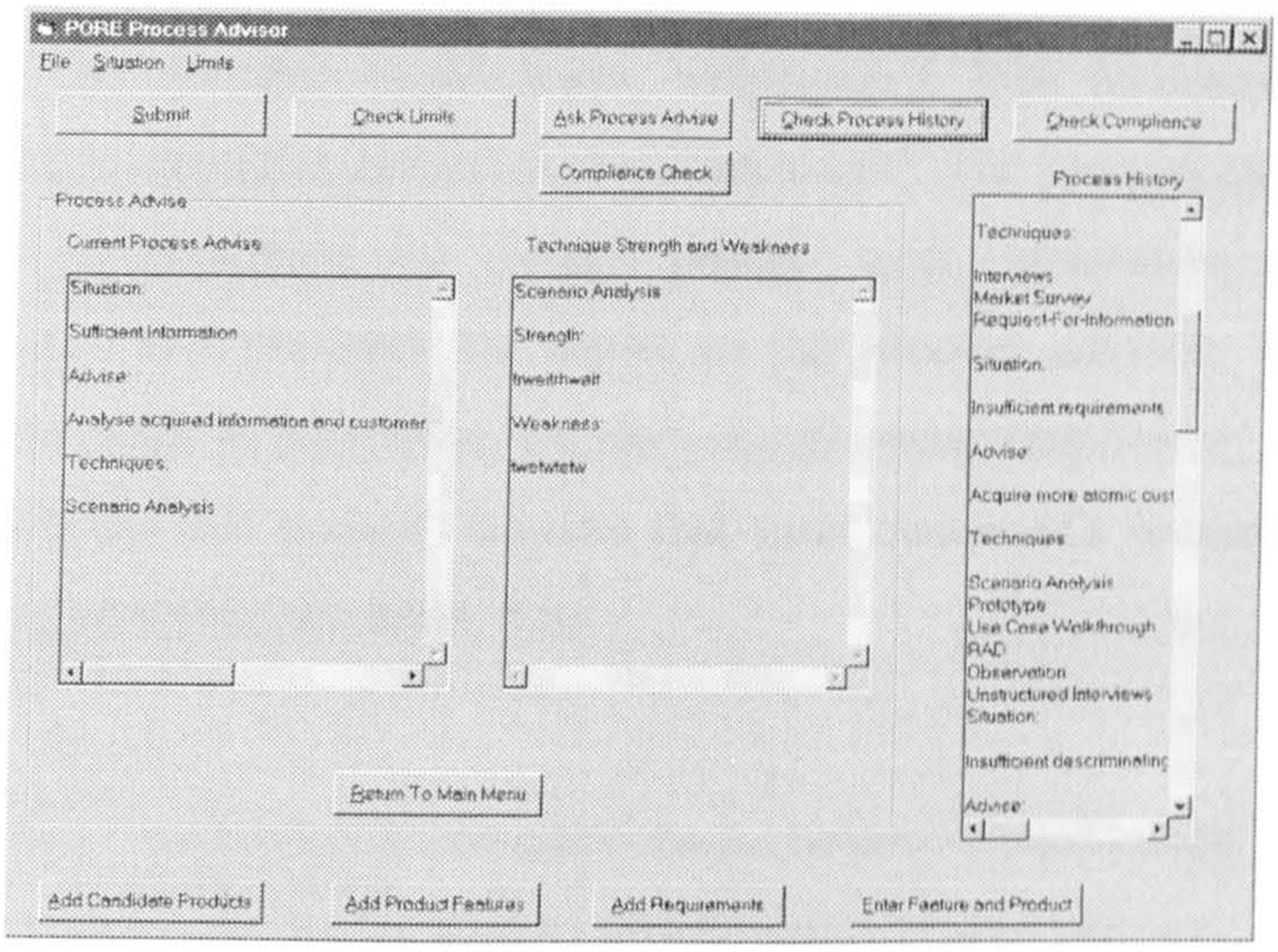
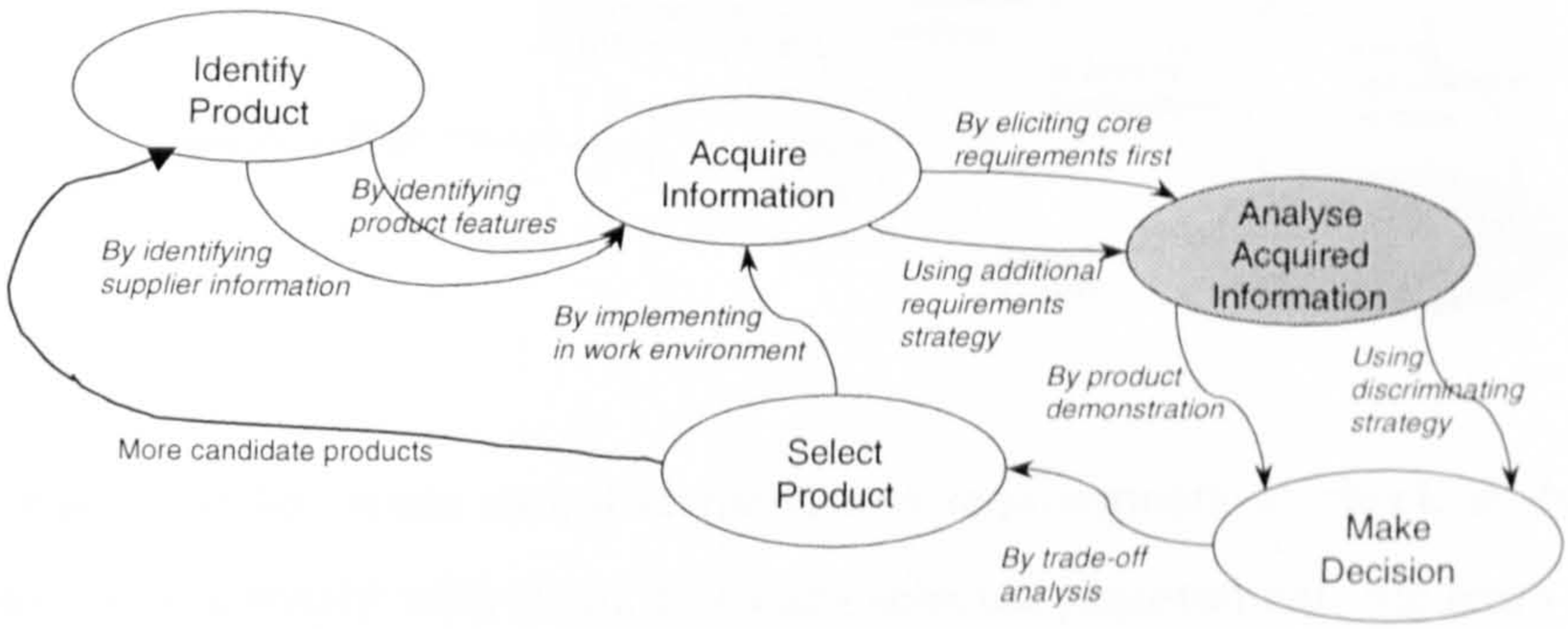


Figure 5.9: The inference engine has detected that there is sufficient information and advises the team to proceed to the next process and analyse the acquired information. The figure also shows the history of the process activities recommended so far.

The requirements engineering team then executes the second generic process of the goal-driven process as shown below:

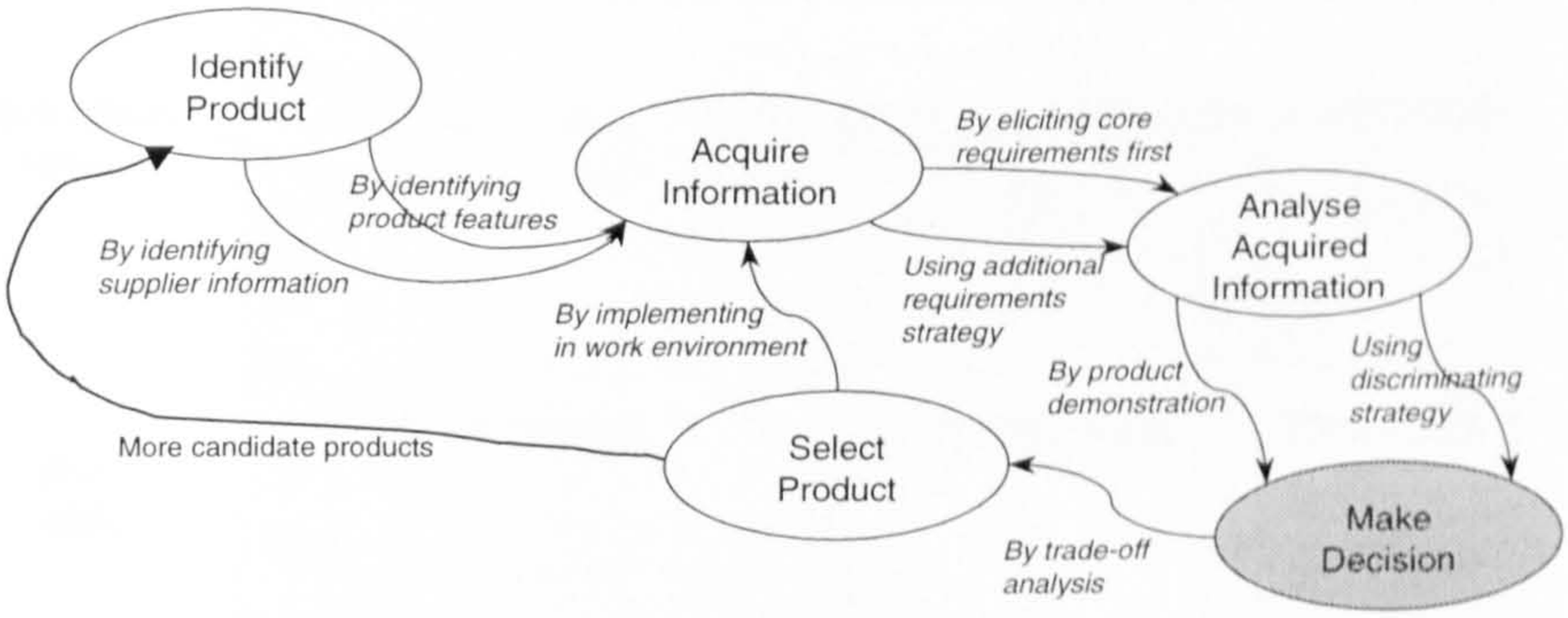


Having followed the advice, the team indicates that it has completed analysing the acquired requirements and product information and asks for further advice and guidance. The process engine determines that the goal has been achieved and checks for current situation. The inference engine fires the situation rule that determines product-requirement compliance mapping.

The situation rule detects that there are no compliance mappings and infers the situation '*no (product-requirement compliance)*' then recommends the techniques *{compliance-checklist, compliance-walkthrough}*. The process engine uses the current situation to determine the current process goal, '*determine product-requirement compliance*' and presents the triplet to the process advisor. The process advisor advises the team to determine '*product-requirement compliance*', that is, to determine those products that possess features that meet customer's requirements. The process engine triggers process chunk 1.8:

```
Process-Chunk:
      Name: 1.8
      Goal: Determine product-requirement-compliance
      Processes: none
      Situation: no(product-requirement compliance)
      Input-information: none
      Techniques: {compliance-checklist, compliance-walkthrough}
End-Process-Chunk
```

The team follows the process advice and executes the third goal-driven generic process as shown below:



The team selected some essential functional requirements to check if there are any products that comply with them. For each selected requirement, the team defined the degree of compliance required based on stakeholder requirements. The inference engine checks for compliance relationships to:

(a) determine products that have features that are mapped to customer requirements;
(b) determine requirements that have no compliance mapping to any product feature;
(c) determine when a product feature has no compliance mapping to customer requirement.

It does this by executing the following situation rules:

```
For Itr = 1 To requirementSelected.Count
      frmProductRequirementCompliance.lisRequireComp.AddItem
      requirementSelected.Item(Itr)
      compScoreSQL = "Select * from compliance where RequirementID in
      (Select RequirementID from requirement where RequirementText =
      '" & frmProductRequirementCompliance.lisRequireComp.List(Itr -
      1) & "')"
Set rstCompScore = dbs.OpenRecordset(compScoreSQL)
Do While Not rstCompScore.EOF
      frmProductRequirementCompliance.lisFeatureComp.AddItem
      rstCompScore.Fields("FeatureName").Value
      frmProductRequirementCompliance.lisProdComp.AddItem
      rstCompScore.Fields("ProductName").Value
      frmProductRequirementCompliance.lisScoreComp.AddItem
      CStr(rstCompScore.Fields("ComplianceScore").Value)
      rstCompScore.MoveNext
Loop
Next
rstCompScore.Close
Exit Sub
```

Figure 5.10 shows the results of executing the above rule and the interaction between the team and the tool to obtain process guidance.

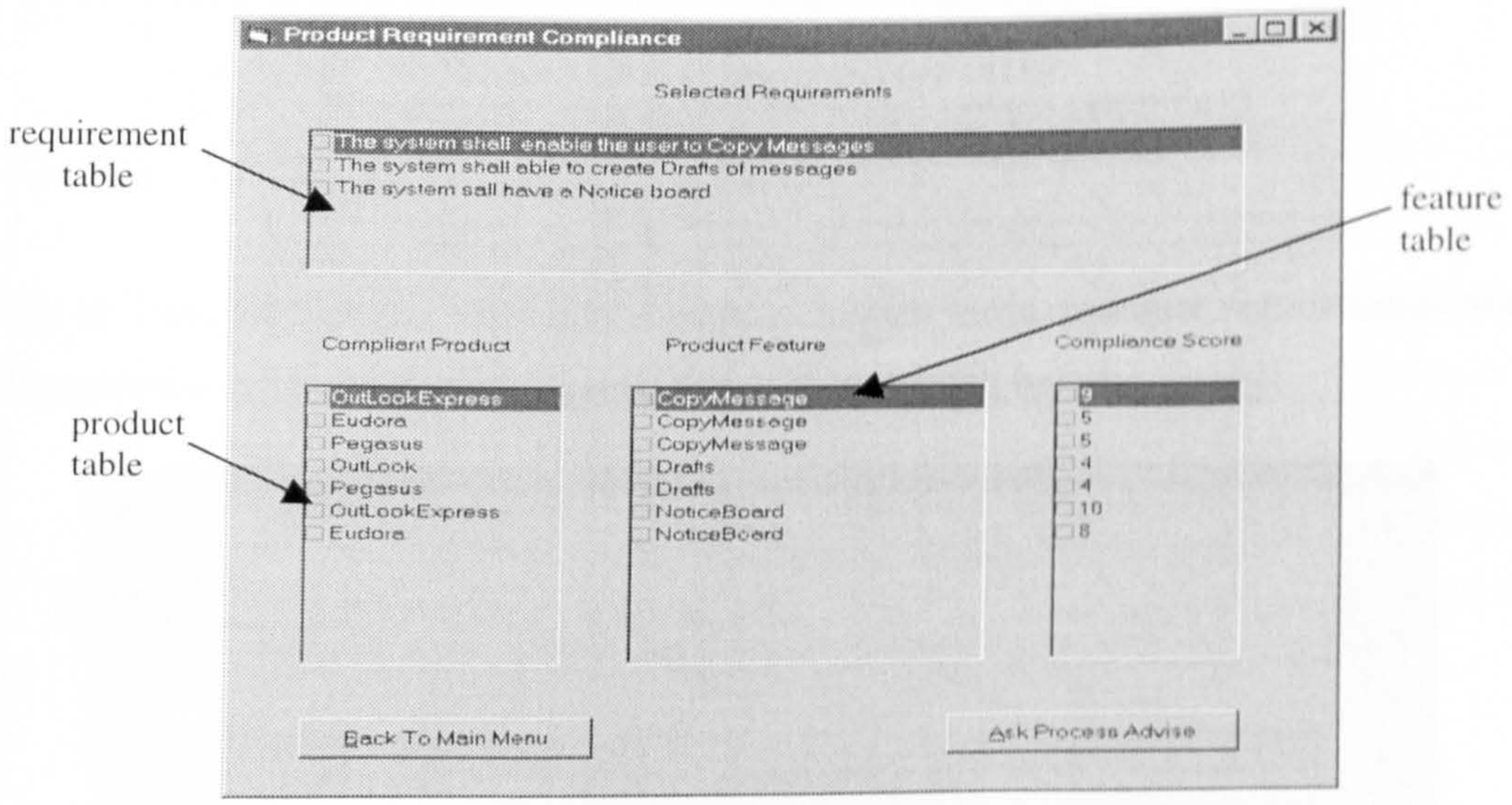


Figure 5.10: Compliance checking scores. The diagram shows that two products meet the requirement to enable copying messages but that OutLookExpress has a higher score than Eudora. It also shows that for creating Drafts of messages two products have equally scores.

After determining compliance relationship mappings, the inference engine determined that for each selected requirement, there is more than one product that has compliance

relationship. In some cases, the products have similar compliance scores. It then determined that there are insufficient requirements that discriminate between candidate products using the following situation rule:

```
If lisProduct.ListCount > 1 Then
      Call produceAdvice("NonDescriminatingRequirements",
      "Insufficient descriminating requirements or product features")
      recProduct2.Close
Exit Sub
End If
```

It then inferred the situation '*no effective discriminating requirements*' and recommended '*card sorts*' as the preferred technique and presented this to the process engine. The process engine used the inferred situation to determine the next process goal '*analyse product-requirement compliance*'. It then presented the triplet to the process advisor and fired process chunk 1.9:

```
Process-Chunk:
      Name: 1.9
      Goal: Analyse product-requirement compliance
      Processes: none
      Situation: no-discriminating(requirements)
      Input-information: content(sub-model(requirements))
      Techniques: {card-sorts}
End-Process-Chunk
```

The process advisor then advised the team to acquire more customer requirements that discriminate between products using card sorts as shown below.

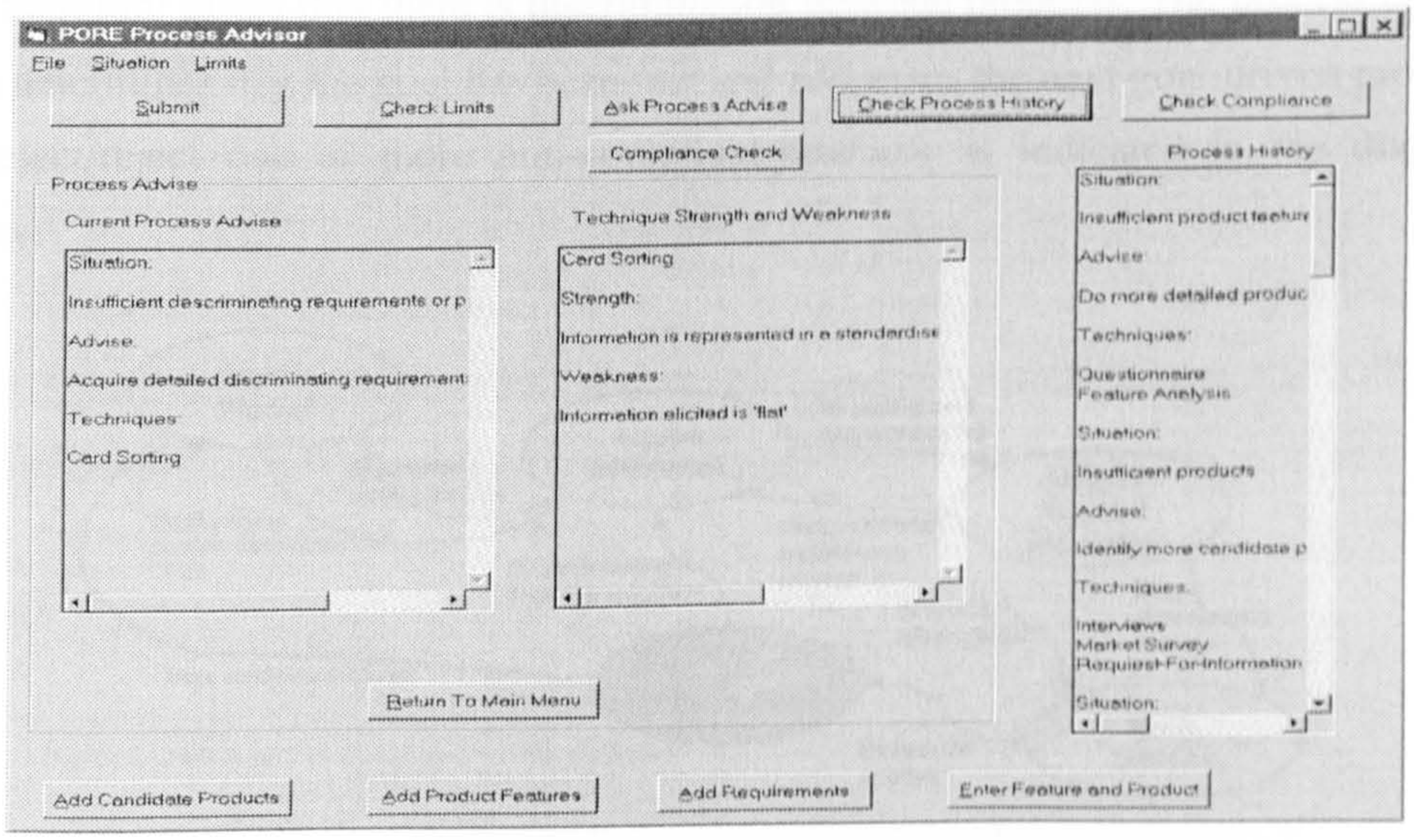
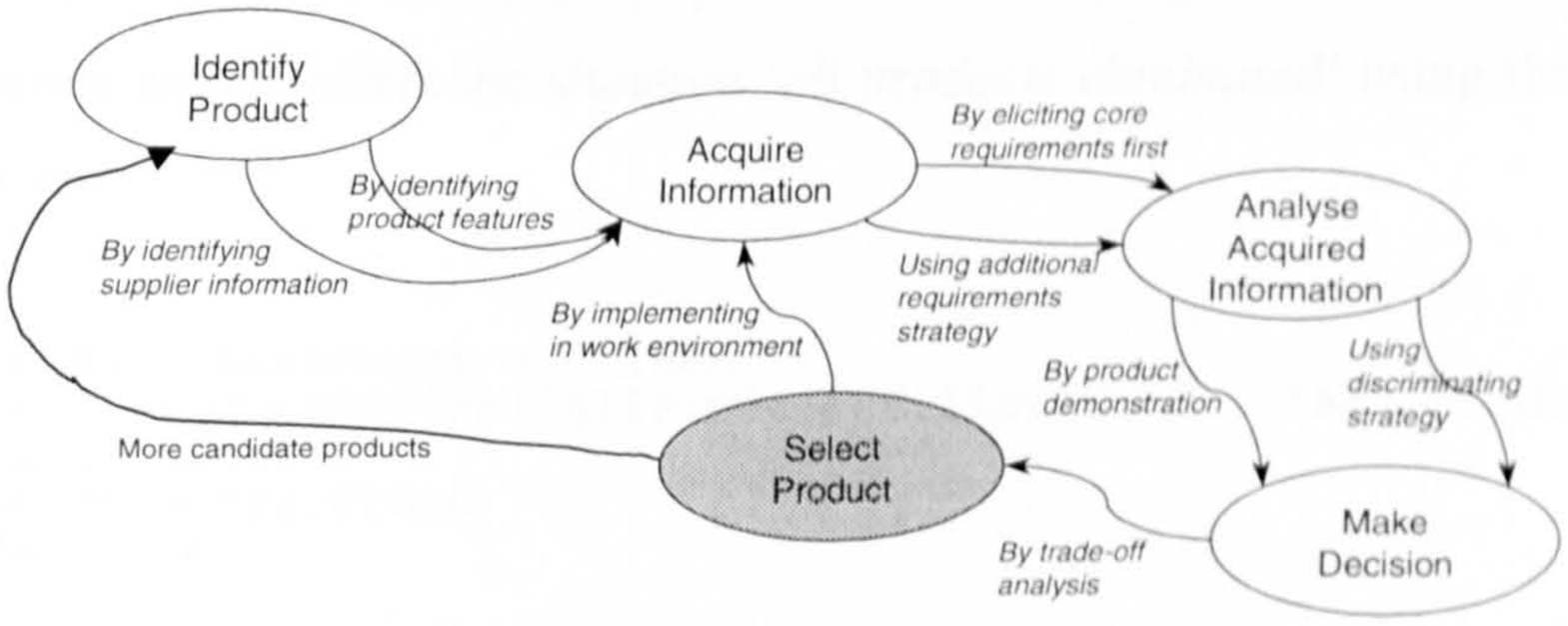


Figure 5.11. Non-discriminating requirements advice as presented to the team. The team is advised to use card sorts as a technique for acquiring discriminating requirements or product information

Using card sorts, a member of the team writes the names of candidate COTS products on 3"x5" cards and asks stakeholders to use the cards to sort the software into categories. Criteria for these sorts (e.g. "compatible with Microsoft™ Word") indicate customer requirements that discriminate between COTS software products. Software categories (e.g. "compatible" and "not compatible") indicate product compliance to these requirements. Card sorts are a very efficient technique in this situation because they only acquire requirements that discriminate between at least two COTS software products. Examples of discriminating customer requirements which are acquired using card sorts include "*the system shall have a UK-English spell checker*", "*the system shall allow the user to set a tailored vacation message*", or "*the system shall allow the user to predefine the font and character size for displayed messages*".

The team followed the advice and acquired sufficient discriminating requirements and entered them in the database, thus changing the state of the requirements sub-model. When the team asked for advice, the inference engine determines that there are no more than one product that has compliance relationship to each essential requirements and therefore infers that there is discrimination between products. The process engine then determines that the goal has been met and addresses the next goal-driven process, to select/reject one or more non-compliant-products as indicated in the diagram below:



When the team asked for process advice, the inference engine inferred the situation that there is a *'small decision space'*, that is the number of requirements, candidate products, product features and compliance relations are small enough to be amenable to decision analysis. The process engine determines the next process goal *'reject one or more non-compliant products'*, presented the triplet to the process advisor and triggers process chunk 1.10:

```
Process-Chunk:
      Name: 1.10
      Goal: Reject one or more non-compliant-products
      Processes:  none
      Situation: small(decision-space)
      Input-information: None
      Technique: {Analytic-Hierarchy-Process, Outranking-process,
      MCDA}
End-Process-Chunk
```

It recommends the set of techniques *{Analytic-Hierarchy-Process, Outranking, MCDA}*. The process advisor advises the team to remove from the list (i.e. reject) those products that do not comply with *'essential atomic functional requirements'*. After this advice, if a suitable product is found the inference engine infers the situation *'Product meets customer requirements'* using the following situation rule:

```
If lisProduct.ListCount = 1 Then
      Call produceAdvice("RecommendProduct", "Product meet customer
      requirements")
      recProduct2.Close
      Exit Sub
End If
```

If a suitable product is not found, the team executes the next process iteration and the first process of the goal-driven generic process is performed again. The iterative process makes it possible that when the goal-driven process to reject one or more non-compliant products is addressed, all products could be rejected. When this happens, the inference engine infers the situation *'all products eliminated'* using the following situation rule:

```
If lisProduct.ListCount = 0 Then
      Call produceAdvice("AllProductsElliminated", "All products
      elliminated")
      recProduct2.Close
      Exit Sub
End If
```

It then recommends technique set *{product-prototype, internet, data-analysis, market-survey}*. The process engine determines the next process-goal '*redo product and requirements analysis or identify new requirements or candidate products*' and triggers process chunk 1.11:

```
Process-Chunk:
      Name: 1.11
      Goal: Redo analysis or identify new candidate products
      Processes:  none
      Situation: all products eliminated
      Input-information: None
      Technique:{product-prototype,internet,data-analysis, market-
      survey}
End-Process-Chunk
```

The process advisor provides the team with the advice to '*redo requirement-product analysis, acquire missing requirements or identify new candidate products*'. The team can relax the strictness of the requirements if possible so that more products will have compliance mapping to requirements. The process advice is presented to the team as shown in Figure 5.12 below:

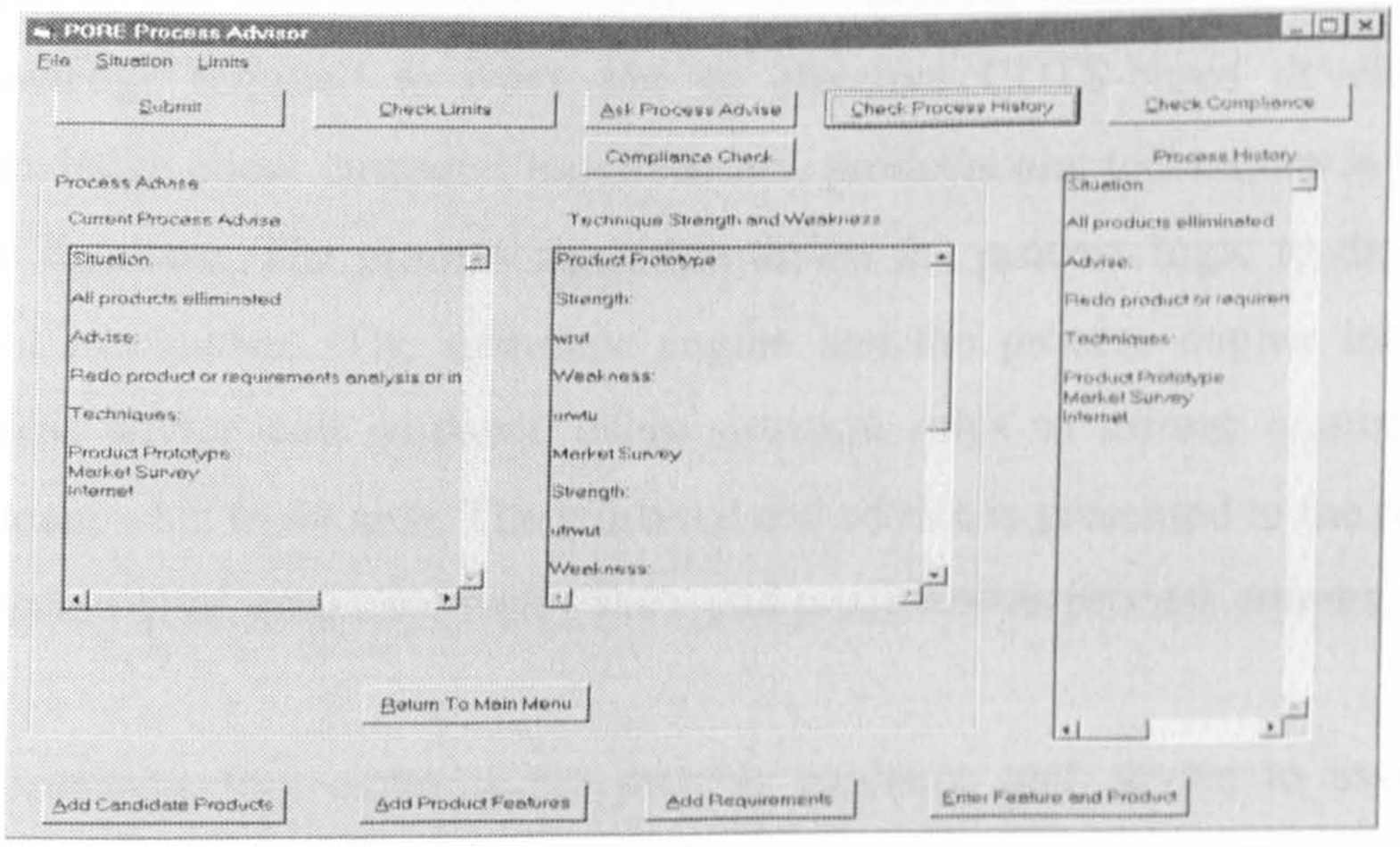


Figure 5.12: All products eliminated. The team is advised to take a number of options, such as either to redo analysis of information, acquire new information or relax some requirements to enable selection.

5.3 Summary

The scenario walkthrough presented above demonstrates how the main components of the theory are linked together by the process engine algorithm and the situation rules which infer properties of the requirement, product and compliance sub-models to drive process guidance. The PORE's process chunks encapsulate knowledge about good and best practice. This knowledge is derived from diverse sources, from textbooks about decision-making techniques to data from interviews with experienced requirements engineers. Situated guidance is needed because the order of processes cannot be predetermined, which in turn depends on what information is available and when. Furthermore, the method box includes techniques drawn from diverse disciplines, to reflect the multi-disciplinary nature of acquiring requirements for COTS software selection.

5.4 Chapter Conclusion.

An overview of the PORE process advisor is presented in this chapter. The process advisor prototype has four main components. The database handles all the problem domain knowledge required to carry out an effective COTS-based development process. Knowledge about customer requirements, products and techniques is held in an integrated database. The process algorithm drives the process logic to direct the computational mechanism. The inference engine and the process engine infer and provide process advice and guidance using situation rules to inform requirements engineering team what to do next. The guidance and advice is presented in the form of a triplet, <*process goal, situation, technique*> and processed as process chunks.

The PORE prototype tool explores the process guidance and advice to assist the requirements engineering team undertaking COTS product evaluation. The tool aims to provide process guidance and advice to the team in a readily understood way by allowing the engineers to ask for or request advice at any stage of the process. A goal-driven process advisor allows the team to select and request advice and guidance on demand. However, the tool has some limitations. For example, it does not yet address issues such as what happens if the combination of the process situation and process goal is equal to zero, how to come out of a situation, what happens if there is no process chunk or does the tool always do what the team want, how do you remove

products from the list so that they are not considered in the next iteration, etc. We aim to solve these limitations and other issues in the future versions of the method.

The effectiveness of the process guidance and advice remains to be evaluated. The following chapter describes a partial evaluation of the PORE method to test its usability, usefulness, and effectiveness. The partial evaluation of the method does not in anyway reflect the author's opinion of the importance of method evaluation. Rather it indicates the limitation of time and resources needed and available for such an evaluation during the research study.

Chapter 6

Evaluating the PORE method

This chapter describes studies carried out to evaluate the PORE method. Three case studies in three different organisations and one controlled study of experts is described to test the usefulness and usability of the method.

Chapter 6

Evaluating the PORE Method

The PORE method was developed to cover a significant set of requirements acquisition and product selection issues for COTS-based systems development. Therefore, only some parts of the method could be evaluated in this thesis. The parts that were evaluated included the first 2 PORE templates and the situated process advice and guidance. The aim was to gather evidence to test hypotheses H3 – H6. The method was evaluated by comparing its predicted aspects with observations made in empirical studies of experts and case studies of the method's use in practice.

6.1 Overview of the empirical studies

The studies provide data about the application of the PORE templates and the usefulness, applicability and repeatability of the PORE method and its processes, advice and guidance. The evaluation was separated into two parts. In part 1, three case studies of PORE's use in real-world COTS product selection exercises were carried out to test hypotheses H3-H5. In part 2, the usefulness, applicability and repeatability of PORE's process guidance by expert evaluation was carried out to test hypothesis H6. Experienced requirements engineering consultants who procure COTS software products were run through example scenarios and asked to comment on what they would do in that situation and what techniques they would use. Their response results were compared with the advice offered by the PORE process.

This chapter is organised as follows. Section 6.2 describes the three case studies carried out in three different organisations, A, B and C. Organisations A and B tested hypotheses H3 and H4. Organisation C tested hypothesis H3 - H5. A brief description of each organisation and their problems is given. Section 6.3 describes experts evaluation of PORE that tests hypothesis H6. Seven representative process chunks that cover the three PORE templates and the whole process are given to experts as situations and the experts are asked to comment on the advice provided. Section 6.4 provides a summary of the PORE method evaluation, as a basis for future research.

6.2 Three Case Studies to Evaluate PORE

Three case studies were carried out to evaluate PORE's templates. These three evaluations were carried out in three different organisations.

Organisation A is an international bank who were purchasing a new dealerboard system. This system provides dealers with an efficient telecommunications system, and was required to have voice-recording capabilities. The system is now used by the dealers to trade through their brokers and banks. For a number of years the bank has implemented its systems using an ad-hoc process without using any methods or following any guidelines. The result of the ad-hoc system implementations was that dealers were using outdated, inflexible systems. The bank were therefore looking to replace the existing dealerboard system with a more advanced, flexible system that will meet their ever-growing needs and needed advice on how to proceed with the selection of the new system.

Organisation B is an international merchant bank who were installing an anti-virus security system at its international head office in London and at branches throughout the world. The bank was experiencing a number of security problems in its operations including:

- files missing from user machines;
- corruption of files and data in transit from its international branches;
- consistent virus being received on its main central server machines.

These problems were further exacerbated by the bank's provision of internet access to all its employees who were then consistently installing unauthorised, infected software on their personal machines or on the bank's network systems. The bank was also continuously receiving e-mail attachments from other organisations which could be virus carriers. The bank was therefore looking to increase their security by implementing a robust anti-virus system that will meet all its security requirements and needed advice about how to proceed.

Organisation C is a Lloyds of London Managing Agent syndicate. The insurance syndicate wanted to purchase a document management system comprising of two main components – a document scanning system and a document management system. At the time the syndicate was using a combination of a paper-based manual system and software systems which were developed in-house. Due to the changes in the insurance market and directives from Lloyds, the syndicate was required to implement an IT system that will directly link to Lloyds' document management systems. System interoperability and integration were therefore key issues. The syndicate needed advice on how to proceed with the selection of the required system.

For ease of reference, each part or item of the PORE template that was applied during the study is indicated as T1.x or T2.x (e.g. T1.2 indicates that item 2 of template 1, i.e. *acquire customer requirements*, was used and T2.1 indicates that item 1 of template 2, i.e. *develop simple working prototype of the required system*, was used).

6.2.1 Evaluation of PORE in Organisation A

An evaluation team was set up to select and recommend a suitable off-the-shelf dealing room system to the bank's management. The team surveyed the market using the internet and trade journals to determine the current state of the dealerboard system market and to identify candidate systems (T1.1). The team also visited other merchant banks that had recently implemented similar systems (T1.1). The team conducted informal interviews with the users in these banks to gather information and problems they had encountered during the selection of their systems. To identify the limitations of the current system and to gain more understanding about how dealers work and use the dealerboard system, the team observed them working. The team also conducted informal interviews with the IT staff to gather their experiences and problems with the current system (T1.2).

During these initial interviews, the team identified the following problems with the current system:

- the current system was not Y2K complaint. Many features and processes of a dealerboard system are date-dependent;
- the current dealerboard stations were desk-mounted and physically bolted onto specially constructed desks. This proved inflexible, impractical, inconvenient and costly to relocate the stations;
- the current system had poor security measures. Dealers were issued with smart cards that are programmed to operate the dealerboard system. The smart cards contain vital information about the organisation's trading and if this information were to fall into the wrong hands or the bank's competitors, it could be harmful to the bank;
- the current system was centrally administered. All the information about each dealer and stockbrokers numbers is programmed into the smart cards. However, if any of the cards needs to be changed or updated, they had to be physically removed and put into a specially designed computer for re-programming;
- the current dealerboard system was not compatible with current technologies;
- the current dealerboard systems was linked to the central switch via dedicated wires with specially designed connectors that only worked with the current system.

The team used information about these problems to identify initial system requirements and to develop a questionnaire to acquire dealerboard systems information (T1.3). The questionnaire was sent to all candidate suppliers to request information about their products (T1.3). The questionnaire stated the purpose of the exercise, closing date for responding and basic information about the organisation and their reason for needing a new system (T1.3).

Eleven suppliers responded to the questionnaire. The team then used the information provided by the suppliers to conduct an initial evaluation (T1.5). Products were assessed on how well they met initial atomic customer requirements (e.g. goal 1). Three products were selected for the more detailed evaluation (T1.5). The three suppliers were asked to demonstrate their products at the bank. This was intended to

allow the suppliers to demonstrate the products in a more realistic environment (T2.6) and to allow stakeholders to attend the demonstration session (T2.2).

To enable an effective product evaluation, customer requirements were ranked using weighted score method (T2.3). The team asked members of the risk management department and the IT support staff to weight requirements according to their perceived importance (T2.3). The team then collated and analysed the scores to provide a final weighted score for each requirement.

Before each demonstration, the team asked the candidate supplier to install a simple working system and to connect it to the organisation's main switching system. This was intended to demonstrate compliance with systems compatibility requirements (T2.1). Two suppliers agreed with this arrangement but the third supplier refused and was subsequently eliminated from the selection process. It took approximately eight weeks for the remaining two suppliers to adequately set up and configure their systems before the demonstrations.

During the demonstrations, stakeholders attended the demonstration sessions but only the evaluation team leader, IT support manager, and heads of the risk management groups were allowed to score products on their compliance with customer requirements (T2.7). However, the stakeholders were allowed to ask questions during the demonstration and they gave significant contributions (T2.2). New requirements, which were not previously identified came up as a result of this (T2.6).

After each product demonstration, the team members collated the product's scores into a single final score and ranked them. In the ranking, product A scored higher than product B, but the difference in the scores was small. After careful consideration of the soft requirements such cost, maintenance, technical support and discussions with the main influential stakeholders, the evaluation team recommended product B as the preferred option. Figure 6.1 below indicate the process activities that were performed and the templates that were applied during these activities.

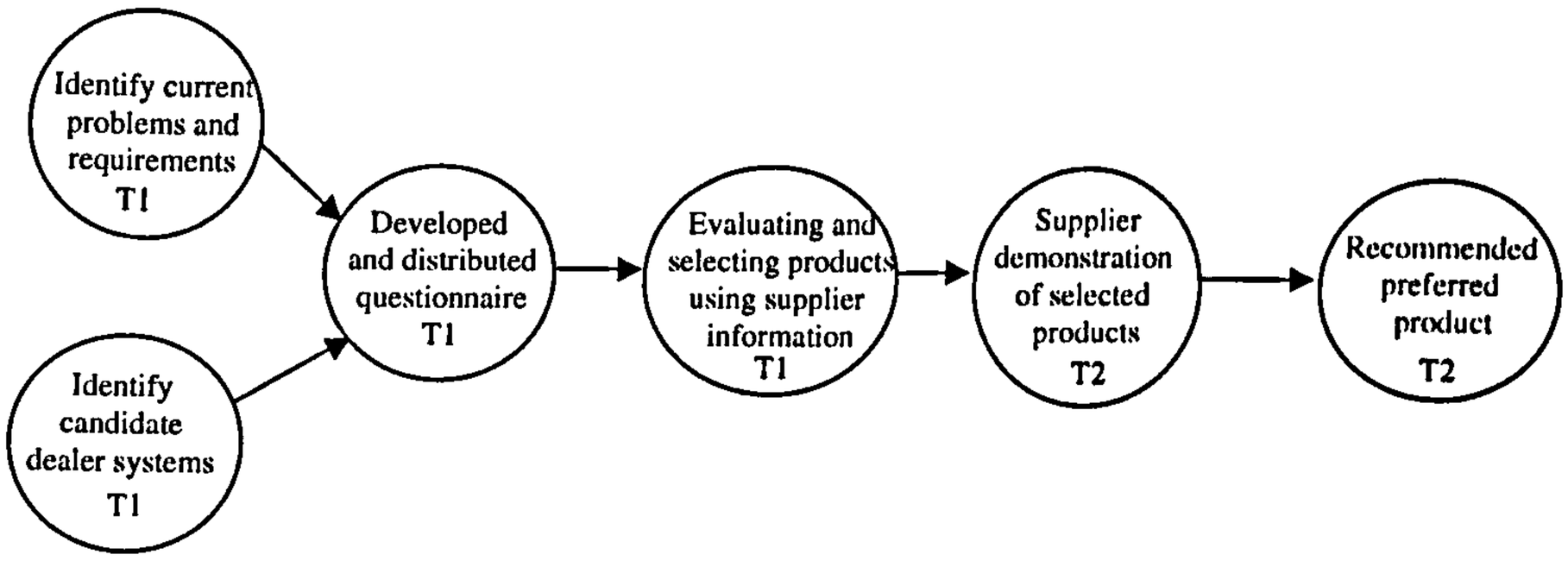


Figure 6.1: Activities performed at each stage of the process, and the PORE templates that were applied at these stages.

6.2.1.1 Results and Lessons Learned

The objective of the case study was to recommend a suitable dealerboard system. A preferred product was recommended, and so the aim of the study was achieved. The PORE templates were applied and found to be useful in providing guidance and advice during the selection process. Techniques recommended in template 1 were very useful in identifying candidate products and acquiring customer requirements. Template 2 provided useful guidance in conducting detailed product evaluation.

Some important lessons were learned about the PORE method:

- Lesson 1 - use weighted scores to evaluate selected products. PORE provided advice and guidance of using weighting methods. Using this guidance, users were asked to rate the importance of their requirements and product attributes. The ratings were then converted into a qualitative weight score for particular features of the dealerboard system;
- Lesson 2- treat the product evaluation as a team effort. PORE advises that stakeholder representatives be present throughout an evaluation to contribute unforeseen requirements and this was useful;
- Lesson 3- there was a clear need for guidance. PORE offered a number of templates to guide the team to acquire requirements and evaluate and select products. The templates were used and were perceived to have greatly aided the study.

The results show that the PORE templates are usable and were used and that the advice and guidance provided is useful.

6.2.2 Evaluation of PORE in Organisation B

To recap, organisation B was an international merchant bank installing an anti-virus security system at its international head office in London and at its branches throughout the world. The bank was experiencing a number of security problems throughout its operations. The bank was therefore looking to increase its security by implementing a robust anti-virus system that will meet all its security requirements and needed advice about how to proceed with purchasing an off-the-shelf system.

An evaluation team was set up to find and recommend a suitable system to the bank management. The team used the PORE templates in the selection of the preferred system. PORE's template 1 guidance was used to identify candidate products by conducting a market survey (T1.1). Techniques that were applied included:

- browsing the internet and sending a general call on it to special interests groups who provided a list of suppliers and users of anti-virus software;
- contacting and visiting customer organisation both within and outside the banking sector who have implemented similar anti-virus systems;
- visiting trade shows and exhibitions such as Infosecurity'97, Networks'97, Secure Computing'97 and Windows NT'97. A large number of anti-virus suppliers were identified at the shows and a large quantity of supplier and product information was gathered;
- reading relevant computing journals. This helped to identify security products that were being advertised and published independent surveys about the capabilities of the current anti-virus products;
- requesting marketing information and product demonstration copies directly from the suppliers.

In parallel to this, the team also identified and acquired core, essential customer requirements (T1.2). Two techniques recommended in PORE's template 1 were used:

- brainstorming – a brainstorming session was conducted with all stakeholders at the bank's London branches. The brainstorming session lasted for about three hours and at the end, a list of essential requirements were identified;
- interviews – after the brainstorming sessions, a number of structured and unstructured interview sessions were held with senior IT officers of the organisation's international division who were responsible for procuring the organisation's software products. Interviews were also held with the IT manager of the London office to determine their local requirements. On average, each interview session lasted for about two hours.

During the product identification phase, a total of 49 products currently available on the market were identified. The first-pass essential customer requirements acquired during the brainstorming and interview sessions were structured into a questionnaire (see Appendix 6a) that was sent either by e-mail, fax or post to all 49 candidate suppliers (T1.3). The questionnaire asked suppliers to indicate the degree of compliance of their products to each of the essential customer requirements. The questionnaire was divided into 5 sections:– basic product information, supplier information, product's requirements coverage, supplier's technical support arrangements, and contract conditions. A covering letter was also sent with the questionnaire. The letter stated the deadline for responses to the questionnaire and explained the process to be used for initial evaluation, and deadlines for being informed of the evaluation team's decision.

Of the 49 questionnaires sent out, only 5 suppliers responded by the deadline time. Using the information provided by the suppliers in their questionnaire responses, (see Appendix 6b) the team evaluated the five products against atomic customer requirements (T1.5). To achieve this, the main stakeholders were asked to prioritise each requirement as essential, desirable or optional. After this paper evaluation, 1 product was rejected and eliminated from the candidate list as it did not comply with most essential customer requirements (T1.9). The four remaining products progressed to second stage of the evaluation – the supplier-led product demonstrations (T2).

To organise and conduct supplier-led product demonstrations, the team used guidance from PORE's template 2. The four suppliers were invited, (see Appendix 6c) to demonstrate their products to the evaluation team. Before each demonstration session, the team developed acceptance test cases (see Appendix 6d) using atomic customer functional requirements, (T2.1). During each product demonstration, the presence of a product feature and that feature's degree of compliance with atomic customer requirements was sought using test cases (T2.5). The team members awarded scores between 0 (not present or non-compliant with customer requirement) and 7 (present and fully compliant with customer requirement) to each required product feature (T2.7). All the demonstration sessions were tape and video recorded (T2.9). PORE's template 2 also recommends that a stakeholder representative be present during all the demonstration sessions (T2.2). However, this was difficult because relevant stakeholders were not always available.

After each product demonstration session, the individual team member's scores were collated and an overall score of the product was produced. To determine the best-fit product, the main stakeholders were asked to rate each requirement in percentages to indicate their importance. After the product demonstrations, the product-requirement compliance scores were then multiplied by the percentage rate score to determine the overall ranking of each product. After producing overall ranking of the products, one product was strongly recommended for trial use within the organisation. The product met most of the organisation's requirements although the other two products had high scores and were recommended as possible alternative solutions. Figure 6.2 shows the activities that were performed and the templates that were applied during these activities.

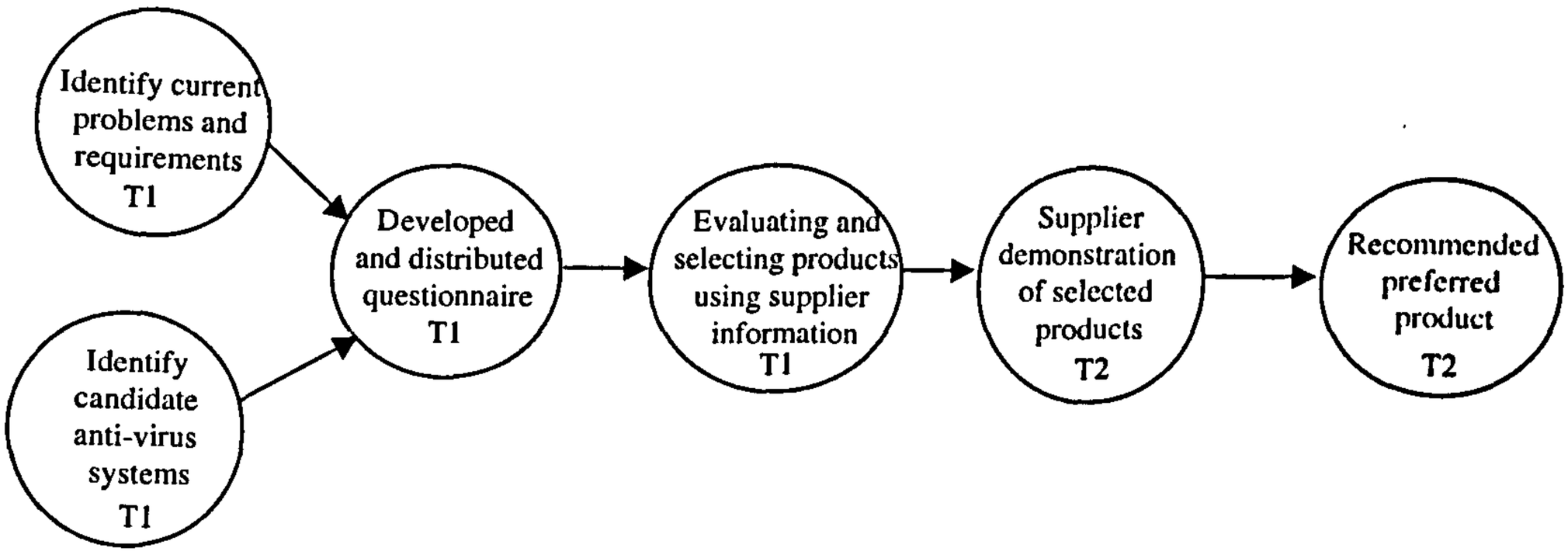


Figure 6.2: Activities performed at each stage of the process and the relevant PORE template that was applied.

6.2.2.1. Results and Lessons learned

The exercise succeeded in recommending a software product to the bank. PORE templates were applied to achieve this success. The templates were very useful in acquiring customer requirements, identifying candidate product and suppliers, gathering supplier and product information, and evaluating and selecting products.

However, the use of the first version of PORE was not completely successful. For example there was a very low response to the questionnaire by the suppliers. PORE does not provide any guidance on what to do if this happens. Other lessons learned about the PORE method are:

- **Lesson-1**: use the questionnaire in combination with other techniques to elicit initial supplier and product information;
- **Lesson-2**: the questionnaire must be short and precise. One of the problems that might have affected the supplier responses is that the questionnaire was too long (15 pages). Suppliers may not have been willing to invest a lot of effort and time without being certain that there will be some benefit to them in the end;
- **Lesson-3**: ask product documentation to be included with the questionnaire response. This will enable the team to compare the results provided on the questionnaire response with actual product descriptions;

- **Lesson-4**: determine the scope and boundary of the product being evaluated. It was very difficult to determine or understand how other systems were associated with the products being evaluated. This made developing test cases very difficult due to the fact that the requirements engineers were required to have knowledge of these systems;
- **Lesson-5:** be aware that not all requirements can be accurately tested due to legal issues. The anti-virus association prohibits anti-virus vendors to deliberately introduce virus to customer machines for the purposes of testing. This made it difficult to test the effectiveness of the products' virus detection capabilities;
- **Lesson-6:** time constraints need to be considered as part of PORE. The time it takes from the start of PORE to finish varies from project to project and the method might be unnecessarily too long for some types of projects. As result PORE should have alternative routes to allow tailoring the method.

In spite of the reported lessons, overall results show that some parts of the PORE templates were applied successfully and the advice and guidance they provided was useful.

6.2.2.2 Discussion

The results of the two reported studies provide support for both the usability and the usefulness of PORE, and indicate possible improvements for the future versions of it. Evidence was found for the two templates, which were found to provide advice and guidance. The selection process was successful. A preferred product was recommended in both cases. Some lessons were learned that perhaps should be incorporated into the method. Partial evidence was found for the predicted use of the recommended techniques. The general sequences of process activities were found to hold and repeatable as expected. Therefore support has been found for hypotheses H3 & H4, stated in the thesis objectives:

> H3: PORE method guidance can be applied in part or in whole to real-world software product selection task;

H4: PORE's effectiveness can form an essential part of a successful product selection task.

Having gained support for the usability and usefulness of the PORE method, the evidence to support PORE's effectiveness is general and weak. The following two studies test for the effectiveness of PORE, beginning with organisation C.

6.2.3 Evaluating PORE in Organisation C

To recap, organisation C was a Lloyds of London Managing Agent syndicate. The insurance syndicate wanted to purchase a document management system comprising of two main components – a document scanning system and a document management system. The syndicate needed advice on how to proceed with detailed COTS software product evaluation and selection.

The requirements engineering team that included the syndicate's IT manager shortlisted the initial candidate products, chosen from the document management system directory. Candidate products were shortlisted using their price range, technical specification, supplier suitability and recommendations from other insurance syndicates. The team also considered the following critical constraints as selection criteria:

- the product's ability to run on the current Windows NT network architecture;
- the product's ability to be customisable using the Visual Basic development toolkit;
- the product's ability to handle Case Processing for the claims department.

The shortlisted suppliers were invited to give product presentations to the evaluation team at the customer site. After each presentation the team realised that more detailed product demonstration against detailed functional requirements and specific selection criteria were needed. The team also realised that they needed to involve stakeholders in the process so as to elicit their specific functional requirements. For process guidance in eliciting stakeholder requirements and for organising the product demonstration, the team applied PORE's template 2 to acquire more detailed

stakeholder requirements and to conduct supplier-led product demonstrations. To test for the presence of support for key predictions stated in hypothesis H5, it was further hypothesised that:

Hypothesis A: the use of scenarios will help in discovering more stakeholder requirements;
Hypothesis B: weighting requirements will help to achieve compliance scores that more closely reflect the customer's critical requirements;
Hypothesis C: having stakeholder representatives present during product demonstration sessions will help in discovering more previously unforeseen requirements.

To test the 3 hypotheses A, B, & C, the team divided the work to be done into two stages – requirements acquisition and product evaluation. Process guidance from template 2 further divided the work to be done into three categories:

- *what to do before the demonstration sessions*;
- *what to do during demonstration sessions;*
- *what to do after demonstration sessions.*

6.2.3.1 What was done before demonstration sessions.

Before the demonstration sessions, the team and stakeholders undertook the following tasks to discover and acquire more requirements:

- developed a simple paper based prototype (T2.1);
- developed scenario models that depict current sequence of tasks performed by the stakeholders that will be affected by the new system (T2.1);
- conducted a series of interviews to identify and acquire stakeholder requirements using identified scenarios.

The first interviews were conducted with the following stakeholders:- two underwriters, the re-insurance assistant, the claims manager and his assistant, the compliance officer and the managing director. Each of these stakeholders described

their current work as a scenario. For each scenario, the sequence steps taken to achieve that scenario were discussed and a first draft of the scenario model was produced. The stakeholders were further asked to brainstorm their work scenarios without describing specific requirements. One example of a first draft of the underwriter's textual scenario model is shown in Figure 6.3.

Underwriter Processes New Risk Scenario

This scenario describes how the underwriter deals with a new risk/proposal from a broker

1 Broker presents underwriter with new risk

2 Underwriter takes a photocopy of the documentation

3 IF risk is to be written THEN
    3.1 Underwriter enters details in the risk system giving risk unique identifier
    3.2 Risk is filed away in current risks file

4 Risk is filed away in metal cases

Figure 6.3. An example of an underwriter's first draft of the scenario model. The scenario model shows the sequence of tasks performed by the underwriter when processing a new risk from the insurance broker.

After the first scenario based interview had identified some initial requirements, the team conducted a second interview in which scenario models were shown to various stakeholder representatives. Each stakeholder representative was asked to walkthrough each scenario and to elaborate the work done. The main aim of the second scenario based interview was to identify and elicit specific requirements and to elaborate the initial requirements that have already been identified.

A third interview was conducted using pictorial storyboards, which depicted the scenarios. The pictorial storyboards show possible ways that the stakeholders would use the future system. Additional new requirements and ideas were elicited. When all relevant scenario storyboards were produced for each scenario, the new elicited requirements were added to the relevant picture/frame as shown in the example in Figure 6.4. Stakeholders were then asked to 'walkthrough' the storyboards to check relevant requirements. More new requirements were identified during the scenario storyboard walkthroughs. Those requirements that the stakeholders agreed with were approved and those that they did not agree with were further investigated. When all stakeholders were satisfied with all the requirements, they were made to sign and

approve the requirements document. The requirements document contained a total of 43 requirements statements.

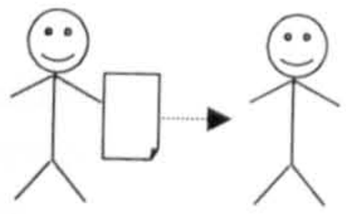
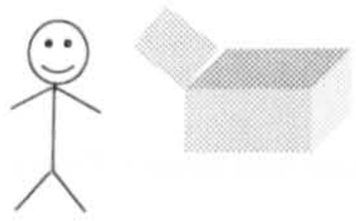
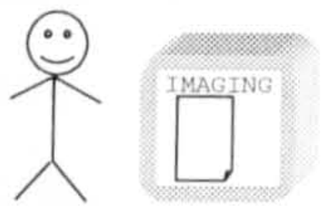
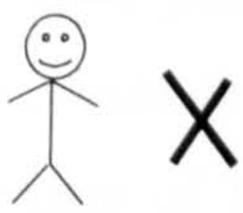
1	2	3	4
			
Renewal of a contract/treaty is received	Underwriter scans contract/treaty renewal	Underwriter indexes the slip by typing in the risk ref no, Year of account, assured/reassured	System gives the option to overwrite the existing slip or create a new version
	R.31.9 The system shall allow the user to attach new pages to an existing imaged document	*R2.9 The system shall display the correct risk slip when its risk reference number, year of account, assured/reassured name are entered using the search facility*	*R.5.9 The system shall allow the user to overwrite/save new version of a risk slip if a risk slip with the same risk reference number exists*
5	6	7	8
			
Underwriter checks image is okay	Underwriter discards the contract		
R.28.9 The system shall force the user to check that an image has scanned properly before allowing them to process a new risk	*R.25.9 The system shall enable a user to view clearly all the text of a single page of a document on a maximised window*		

Figure 6.4. An example of a storyboard showing a stakeholder scenario. Existing requirements are attached to relevant frames of the storyboard.

After identifying and eliciting the core customer requirements, the team asked the stakeholders to weight them (T2.2). A fourth interview was then conducted in which stakeholders were asked to group, rank and prioritise their requirements and then place each requirement in one of four categories:

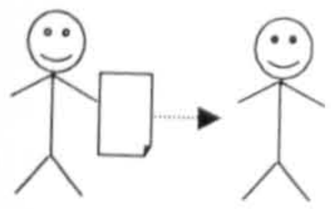
- Essential and mandatory – Very High Priority;
- Important – High Priority;
- Would Be Nice – Medium Priority;
- Don't Care – Low Priority.

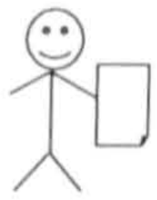
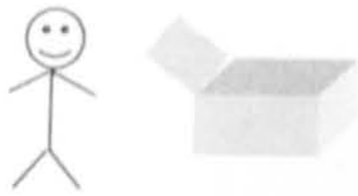
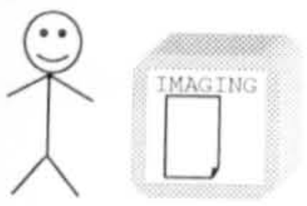
After this, the team asked the stakeholder to rank each requirement within each group and category. The VOLERE method requirement template (Robertson, 1998) was

used to provide measurable fit criteria for each requirement (T2.5). Some functional requirements were iteratively refined into simple atomic statement to ensure that no requirement contained any AND/OR statements. One such refinement example is shown below:

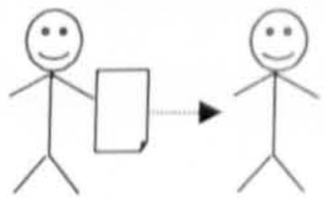
Identifier *R5.9.1*
Description *The system shall warn the user before overwriting risk slips*
Type *Functional*
Priority *High*
Source *Andy (Underwriters)*
Scenario *Underwriter renews Contract/Treat*
Measurable fit Criteria: *The system shall allow the user to overwrite a risk slip if a risk slip with the same risk reference number exists*

After the requirements were weighted, ranked and prioritised, the team used them to generate test cases for detailed product evaluation. The test cases were criterion by which the team judged the presence or otherwise of a required feature in each product and the degree of compliance of that feature to a customer requirement. Scenario models were used to design the test cases and structure the evaluation sessions. The evaluation questions were formulated around the scenarios by combining the scenario models and customer requirements in the test cases. An example test case is shown in Figure 6.5. While the test cases were being developed, shortlisted suppliers were informed by telephone, then by formal letter of the aims and format of the evaluation sessions.

<table>
<tr>
<td>1
 
Broker gives risk slip to the underwiter</td>
<td>2
 

Underwriter enters details from risk slip into risk system which generates unique risk ref. number</td>
<td>3
 
Underwriter stamps risk slip, enters risk ref. number, signs and dates it.</td>
<td>4
 
Underwriter scans the risk slip</td>
</tr>
<tr>
<td></td><td></td><td></td><td></td>
</tr>
<tr>
<td>5
 

Underwriter indexes the slip by typing in the risk ref number, year of account,assured/reassured</td>
<td>6
 

Underwriter checks image before processing new risk</td>
<td>7
 
Underwriter returns risk slip to Broker</td>
<td>8</td>
</tr>
</table>

ReqID	Requirement Description	Compliance Score	Comment
R2.9.1	The system shall index a risk slip by risk reference number, year of account and assured/reassured	0-1-2-3-4-5	
R32.9.1	The system shall ask user what type of document has been scanned before saving the image, e.g. Treaty, Slip, Contract	0-1-2-3-4-5	
R32.9.2	The system shall store the image in the correct directory depending on the response from the user in R32.9.1	0-1-2-3-4-5	
R17.9	The system shall force the user to check that an image has been scanned properly before allowing them to process a new risk	0-1-2-3-4-5	
R31.9	The system shall allow the user to attach new pages to an existing imaged document	0-1-2-3-4-5	

Figure 6.5. Scenario-based test case generation for the underwriter's new risk scenario. The table shows the compliance scores of between 0 – 5, with 0 = no compliance and 5 = total compliance

Figure 6.6 shows the activities that were performed at this stage.

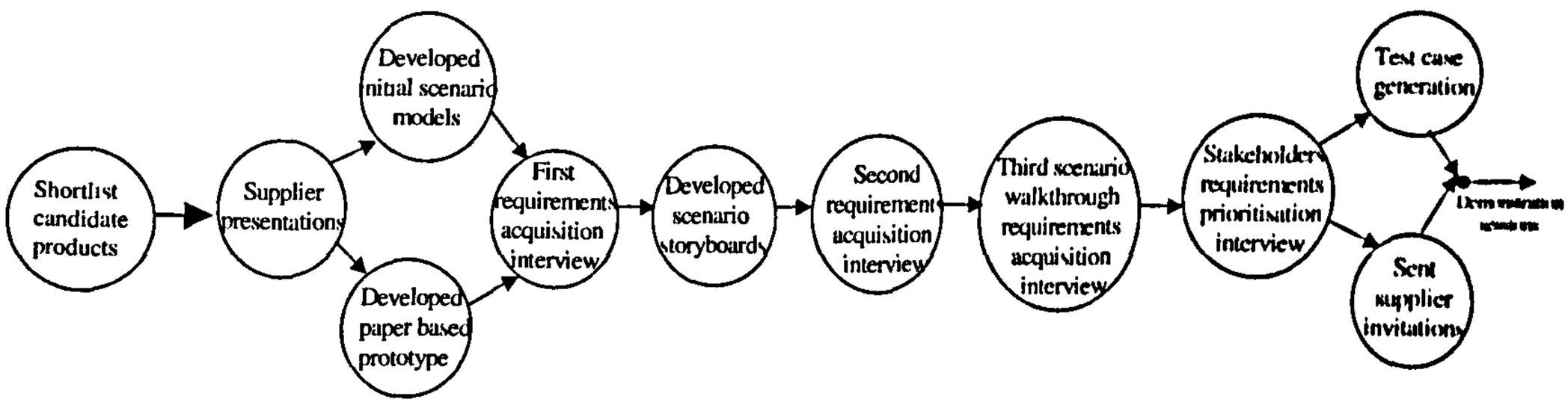


Figure 6.6. The activities performed before product demonstration sessions. All activities involved both the evaluation team and relevant stakeholders.

6.2.3.2. What was done during the demonstration sessions.

Three suppliers were invited to give product demonstrations. Each evaluation session was planned to last for 2 hours. Of the stakeholders that were interviewed, three were nominated as representatives of their respective business sections - underwriting, claims and re-insurance – and were asked to attend each supplier demonstration session (T2.2). However, several other stakeholders attended the demonstration sessions partly, out of curiosity and partly because they were encouraged by the management as a way of selling the new initiative to them to overcome resistance to the new system. During the demonstrations, only the three members of the evaluation team scored the products. Each team member scored the product independently using the test cases. One team member was chosen to lead the evaluation session during the demonstrations. Each member awarded a score of between 0 and 5 depending on whether the feature was demonstrated and the degree it complied with the customer requirement (T2.7, also see Appendix 6e). Only the three evaluation team members were allowed to ask questions about the product during the demonstration (T2.6) but stakeholder representatives were consulted to provide domain specific information and clarification (T2.2). A scribe was assigned to record all the key decisions and some questions that arose during the demonstration (T2.9).

6.2.3.3. What was done after demonstration sessions.

At the end of each demonstration session, the three evaluation team members agreed a final score for each product's compliance to each requirement. When there were disagreements, each member explained their reason for awarding the score. The recordings made by scribe were also consulted for clarification on some of the decisions. After this process, final agreed scores for the product were produced and entered into a spreadsheet to calculate the product's overall ranking. From the ranking, the team judged and agreed that although all products met the customer's requirements, one product ranked higher than most. From this information, the team made the following recommendations:

- that the syndicate must obtain working copies of the first two products and install them in their working environment for a limited period (e.g. 6 weeks). This would enable prospective users to familiarise with the product for further evaluation and to discover new requirements (template T3). This would also enable the users to judge which product performs better in a realistic business environment;
- that once the products have been installed, the IT manager must obtain feedback from users to further elicit new requirements (T3);
- that during this period, much attention should be paid to non-functional requirements such as integration requirements, interoperability requirements, (since the chosen system will be integrated with Lloyds of London IT systems), training requirements, usability requirements, interface requirements, training support, costs and quality of provided (since this is a new system unfamiliar to most users, T3);
- that legal advice must be sought for negotiating a contract with the final chosen supplier. The contract should be based on stakeholders' requirements;
- that all stakeholders who attended the demonstration sessions were further interviewed to identify further requirements (T2.10).

At the end of the selection process, the team interviewed and sent a questionnaire to five key stakeholders that were involved in the process to get feedback about the selection process. The stakeholders were asked eight specific questions that were designed to test the effectiveness of PORE's template 2 (see Appendix 6f). The results

of the interview and questionnaire are summarised below. Appendix 6f presents the detailed results from the questionnaire. The process activities that were performed during this stage are shown in Figure 6.7 below:

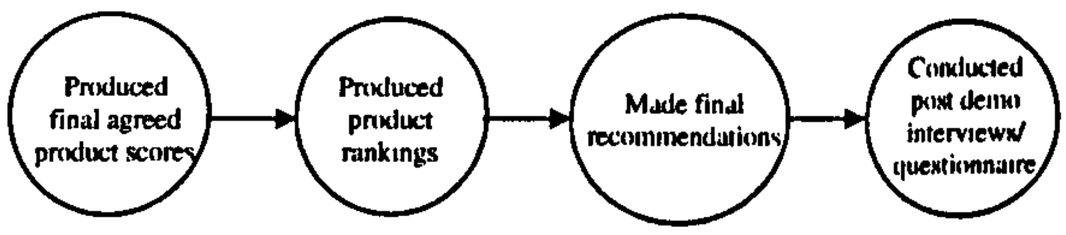


Figure 6.7. Process activities performed after product demonstration sessions

6.2.3.4. Results

All five stakeholders who were sent the questionnaire and interviewed indicated that the product demonstration sessions gave them more knowledge about the capabilities of document management product that they previously did not have, (see Appendix 6g). Specifically, the stakeholders indicated the following key benefits:

- four stakeholders indicated that:

(i) the scenario based interviews were useful in helping them identify most of their requirements;

(ii) use of scenario models to breakdown requirements by task was very useful;

(iii) scenario models helped to ensure a common understanding of requirements between the requirements engineer and themselves;

- four stakeholders indicated that product demonstrations were informative and quiet helpful;
- four stakeholders indicated that they felt more involved in the selection process than in any other project they have previously done;
- all five stakeholders indicated that the method provided an excellent way of systematically filtering out the non-compliant products;
- all five stakeholders indicated that they would use PORE again if they were to select a product in the future.

These results therefore, strongly support hypotheses A and C:

Hypothesis A: the use of scenarios will help in discovering more stakeholder requirements;

Hypothesis C: having stakeholder representative present during product demonstration sessions will help in discovering more previously unforeseen requirements.

Evidence to support *Hypothesis B: weighting requirements will help to achieve compliance scores that more closely reflect the customer's critical requirements,* although present, its weaker. However, the stakeholders also provided a number of potential shortfalls of the method and some suggestions, (see Appendix 6g):

- four stakeholder indicated that the demonstrations sessions were too long and not well organised;
- two stakeholders indicated that the questioning of suppliers by the evaluation team was too aggressive and intimidating;
- two stakeholders indicated that the process was too long and a long period of time elapsed between the starting and finishing the project;
- one stakeholder indicated that all product demonstrations should have been done within the same week.

Most stakeholders commented that the structure and organisation of the evaluation sessions was not properly organised. Particularly, they commented on the fact that some requirements were tested several times during the evaluation sessions and this created unnecessary excessive pressure on the product demonstrator. For example, the demonstrators were frequently asked to scan a document, index it using a reference number provided by the evaluators and then asked to retrieve that same document. On more than one occasion, the demonstrator was unable to perform this simple task due to either technical incompetence or technical problems and this led to very long periods of waiting in silence. Although these requirements were critical, the stakeholders felt that they slowed the evaluation process and felt that they could have been tested at a later stage using evaluation copies of the software and the focus during the evaluation sessions should have been on core functional requirements.

However, these faults were due to the way the team applied the method, not due to a weakness in the method itself.

Stakeholders also suggested some improvements to template 2. Particularly, they felt that it would be helpful if both the stakeholders and product demonstrators have a product familiarisation session before the formal evaluation sessions. The familiarisation sessions should involve the evaluation team members, all stakeholders and suppliers and would provide:

- an opportunity for questions and answers between stakeholders and suppliers without the problem of disrupting the formal evaluation;
- an assessment of the generic capabilities of the product and for the evaluation team to familiarise with the product before formal evaluation;
- a vehicle for new requirements that help distinguish between candidate product which can then be tested during formal evaluation. The formal evaluation will then focus on evaluating those requirements that help discriminate between products.

The results of the reported study provide evidence for the usefulness of PORE guidance and indicate possible improvements for the future versions of it. Evidence was found for template 2's guidance which was perceived to be useful and beneficial by all stakeholders who were involved in the study. Some lessons were learned that perhaps should be incorporated into the method. Partial evidence was found for the predicted use of the recommended techniques. The general sequence of process activities was found to hold and repeatable as expected. Therefore support has been found for hypotheses H5:

Hypothesis 5: PORE guidance is perceived to be useful and usable by people involved in the product selection task.

6.3 Expert Evaluation of PORE

6.3.1 Study Method

The PORE method, process guidance and advice were also tested using expert evaluation to obtain direct feedback on its effectiveness, usefulness, repeatability and applicability in different situations. Seven focused requirement acquisitions and product selection scenarios were selected for evaluating process guidance and advice. The scenarios provided descriptions of product selection 'situations' that can arise during the process. The situations were described in sufficient detail to allow product evaluation implications to be inferred and reasoned about, (see Appendix 6g). Table 6.1 lists the seven situations that were used.

Situation	Situation Chunk
No customer requirements	Process-Chunk 1.2: This chunk determines that there are no requirements and advises the team to acquire customer requirements and recommends relevant techniques to use
No candidate products	Process-Chunk 1.3: this chunk advises the team to identify candidate products existing in the market and recommends suitable techniques
Insufficient product and supplier information	Process-Chunk 1.4: this chunk advises the team to acquire information about identified candidate products and their suppliers. It recommends techniques to use.
Non or insufficient discriminating requirements	Process-Chunk 1.5: this chunk determines that there are insufficient requirements and advises the team to acquire more discriminating requirements and recommends relevant techniques
A large number of products identified, reduce products to a manageable list	Process-Chunk 1.8: this chunk advises the team to reject least compliant products using information provided by suppliers so as to reduce the number to a manageable list.
Evaluate in detail shortlisted products	Process-Chunk 1.9: this chunk advises the team to evaluating in detail the shortlisted products by conduct supplier-led product demonstrations
Decide and select final preferred products from the shortlisted ones	Process-Chunk 1.10: this chunk advises the team to reject non-compliant products from supplier-led demonstrations

Table 6.1: The seven situations used in the evaluation study

Three situations described acquiring customer requirements from stakeholders and product information from suppliers. One situation described the identification of candidate products that are currently existing in the market. Three more situations

described the evaluation and selection of products that comply with customer requirements. The seven situations were chosen so that they would provide coverage of all parts of the method. The full description of the situations is given in Appendix 6h.

Two requirements engineering consultants from two different organisations took part in the evaluation. Before hand, both completed questionnaires (see Appendix 6g) about their background. Results are described in table 6.2. One engineer had 30 years of experience in requirements engineering, the other had 8.5 years. They have experience of requirements engineering and COTS product selection in a variety of application domains.

Requirements Engineer	**Expert 1**	**Expert 2**
Experience		
Years in requirements engineering	30	8.5
Requirements elicitation and acquisition	√	√
Requirements modelling	√	
Requirements measurement	√	
Requirements weighting	√	
Requirements engineering techniques and methods	√	√
Product & Supplier Evaluation		
COTS software product identification techniques	√	√
COTS software product evaluation and selection	√	√
Supplier evaluation	√	√
Experienced in customer representation during product evaluation		
Software procurement process	√	
Decision-Making Analysis		
Decision-making in software product evaluation	√	
Decision-making techniques	√	√
Decision-making tools		

Table 6.2: The background of the two requirements engineers. The symbol "√" indicates that the requirements engineer has the relevant experience.

The engineers were given a brief overview description of the research and the high-level PORE process for selecting a COTS software product. They were also given a

description of the scenario that was described in chapter 3 to select and recommend an office e-mail system. The engineers were then given the product selection situations one by one and asked to describe what they would do in that situation, and what techniques they would use. While they were describing their solutions, their conversations were audio taped for reference. After all the situations had been completed, the engineers were asked to comment on the high-level PORE process that they were shown at the beginning.

6.3.2. Results

Table 6.3 gives the solution provided by each engineer for each situation (see Appendices 6i & 6j). The table also shows the method's predicted solution to each situation, which is compared to the solution provided by each requirements engineer.

Situation	Theory Prediction	Expert 1	Expert 2
At the beginning of the process, the customer has expressed the need to purchase one or more off-the-shelf solutions. However, there are no customer requirements identified so far that the e-mail package should meet. Some stakeholders are familiar with some e-mail packages. So what would be your next objective and what techniques would you use to reach this objective?	(1) acquire customer requirements (2) Techniques: brainstorming, use cases, interviews	(1) find out why they are doing the project in the first place and quantify it; (2) use requirements meta-model (3) start with project blast off to help (4) look at the first eight items of the requirements template	(1) find out why customer need to purchase the system or product (2) identify the stakeholders (3) identify what capabilities you want in the package (4) identify high level requirements (5) think about high level IT architecture and infrastructure (6) determine what packages are available
At this point in the process, you have not identified any candidate products that would meet customer requirements. However, some stakeholders suggested 3 possible e-mail packages available. So what would be your next objective and what techniques would you use to reach this objective?	(1) identify candidate products (2) Techniques: internet, market survey, vendor conferences, other organisations	(1) identify candidate products (2) Techniques: internet, trade papers, other organisations, market survey	(1) have a pure requirements specification before identifying the capabilities to avoid bias into products (2) go to e-mail vendor conference to identify products (3) visit vendors that have been shortlisted (4) talk to other users (5) get the users to try the product (6) don't force people to use the product (7) use prototype as a technique (8) involve users in the selection process
You have now identified 30 candidate products in the market. Access to these products and their suppliers is	(1) acquire product and supplier information (2) Techniques: questionnaire, request-	(1) use what the web has told you (2) down load product summaries from the web	(1) list requirements in priority (2) get marketing literature

excellent. Furthermore during the requirements acquisition process, over 45 essential requirements have been elicited. However, you have insufficient product and supplier information. So what would be your next objective and what techniques would you use to reach this objective?	for-information	(3) send requests to the companies (4) talk to other people send questionnaire	(3) get vendor literature (4) match capabilities to customer requirements (5) make sure that users are involved (6) shortlist on user requirements
At this stage in the process you have identified that a large number of the 30 candidate products offer similar functionality to meet the office employees' requirements. For example all the e-mail packages meet the requirements 'the system shall send messages' and 'the system shall receive messages'. The problem is that these requirements do not help you to discriminate between the packages. So what would be your next objective and what techniques would you use to reach this objective?	(1) Acquire discriminating atomic requirements (2) Technique: Card sorts	(1) apply our idea of fit criteria, i.e. if there is customer requirement 'send message' you determine if product sends message according to the customer requirement	(1) not all products will meet requirements exactly the same (2) determine how well each product meets the requirements (3) determine how well each product meet the core requirements (4) use other requirements like cost, customer base to discriminate
Because of time pressure, you are unable to examine in detail all of the 30 candidate products you have identified. How would you reduce the number to a manageable list and on what basis would you base your decision to remove the product from the list. What would be your next objective and what techniques would you use to reach this objective?	(1) Reject least compliant products using supplier information (2) Technique: use core atomic functional requirements	(1) pick 10 key requirements and use them to eliminate the products	(1) match the requirements and use compliance scores to eliminate the product from the list (2) have a structured way of dealing with product elimination (3) record why you took that decision and at what point so that you can go back to it later if need be (4) have a group decision and let everybody sign up to it
At this stage in the process, 6 candidate products have been shortlisted. The process, which led to short-listing these products, was trouble free although some suppliers who were not shortlisted were not happy with their rejection. How would you conduct evaluation sessions as a basis for selecting a final preferred product from the 6 shortlisted? What would be your next objective and what techniques would you use to reach this objective?	(1) conduct supplier-led demonstration sessions (2) Technique: test cases, compliance scores,	(1) do the products satisfy the requirements equally? (2) use ordering for short listing (3) involve stakeholders	(1) do more detailed analysis (2) get people to try out or use the product (3) have a structured process in advance on how you are going to do the selection (4) have a vendor demonstration (5) use demos for short listing (6) consider non-technical issues as well
At this stage of the process, you have conducted the evaluation sessions of the 6 shortlisted products. How would you decide which products to reject from the	(1) Reject non-compliant product(s) from supplier-led evaluation demonstrations (2) Techniques:	(1) use stake holders to priorities the products	(1) use differentiation factors between the requirements (2) base it on how your understanding (3) use soft issues like the

short-list and on what basis would you select the final product(s)? What would be your next objective and what techniques would you use to reach this objective?	Decision-making techniques		supplier capability (4) get stakeholder representatives to brainstorm preferred products (5) stakeholders to decide which one to select

Table 6.3: The expert knowledge provided by each engineer for each situation.

To determine the fit between the experts' advice and PORE's own process guidance, the following five degrees of fit were applied between the guidance and either expert's advice.

Condition A – expert-PORE advice matches iff the expert provided information which exactly matches PORE's process advice and technique(s);
Condition B – expert-PORE advice mismatch iff the information provided by the expert does not match PORE's process advice and techniques;
Condition C – equivalent match occurs iff the information provided by the expert is an analogous substitution for the process advice and techniques provided by PORE;
Condition D – expert-PORE match but the matching is too course, i.e. the expert's advice is too general but covers that provided in PORE;
Condition E – expert-PORE match but the matching is to narrow, i.e. the advice provided by the expert does not include all the advice provided in PORE.

Table 6.4 shows the application of the five degrees of fit to PORE and the 2 experts' advice. For each situation, the expert indicates what advice they would recommend, i.e. what they would do next when in that situation and what technique they would choose to use in that situation.

Situations	Expert 1		Expert 2	
	Advice	Technique	Advice	Technique
Situation 1	C, D	B	C, D	B
Situation 2	A	A	C, D	A,
Situation 3	C, E	A	C, D	C, D
Situation 4	B	B	B	B
Situation 5	A, E	A, E	C, D	C, D
Situation 6	C, D	C, D	C, D	C, D
Situation 7	B	B	C, D	B

Table 6.4: Analysis of fit between process advice from PORE and two experienced requirements engineers.

The matching of predicted process advice with that provided by experts was investigated further. There were two occasions in which the experts' advice matched exactly with that given by PORE, three occasions in which advice did not match at all and in the remaining situations the experts gave equivalent advice. Results show that most advice given by experts was more general and is included in the advice provided by PORE. The number of predicted advice matched to that given by experts is shown in table 6.5 below. The results show that the method predicts most advice and guidance that the experts would provide in similar situations.

Predicted Match	Frequency
AA	2
BB	3
CDCD	4
CDA	1
CDE	1
CDB	3
TOTAL	14

Table 6.5: The number of times the advice given by the experts matched with advice that PORE gives.

The following sections give brief descriptions of the expert's comments for each situation (see Appendices 6i & 6j).

6.3.4 Summary of the results

Situation 1: no customer requirements identified – *both experts gave equivalent match (C) but course process advice (D) and mismatch techniques (B).*

E1 recommended first of all to determine the purpose of the project. This helps to "identify the boundaries and scope of the system". The expert went on to say that

"without knowing the reason for the system it makes it very difficult to figure out which products should be identified and to be included in the candidate list". For this, E1 recommends to use the idea of 'project blast off', which is a "joint application development meeting where the stakeholders gather enough facts to ensure that the project has a worthwhile objective, is possible to achieve and has commitment from the principal stakeholders". The expert did not recommend any other specific techniques for acquiring the stakeholders initial requirements.

E2 recommended that the first thing to do is to: (1) identify "why the customer wants to go for the off-the-shelf solution in the first place"; (2) identify "everybody that has a stake in the system and what are the requirements for the customer who is actually paying for the system"; (3) identify "people who have been using the old system and elicit their problems and frustrations".

From this, E2 suggests:

- getting a high level understanding of the needs before starting to look for candidate products and their capabilities;
- determining what the existing IT infrastructure and what are the plans for upgrading it;
- determining high level architecture and an understanding on how that infrastructure is going to develop over the next following few years.

Once this has been done, the expert suggests to:

- look at what packages are available out in the market;
- identify the package attributes and try to match them to the initial requirements elicited from the different stakeholders that are going to use the package.

The expert does not suggest any specific techniques to use at this stage but recommends looking at or talking to other people who have used similar packages before.

Situation 2: no candidate products yet identified – *E1 provided information that exactly matched PORE's process advice and techniques (A), but E2 provided equivalent (C) but course process advice (D) and mismatch techniques (B).*

E1 'if the candidates products are not yet known, the internet is the obvious first place for find out about what products are available in the market'. The expert went on to say that before the internet it used to take "*a lot longer to identify existing products because you used to go to trade papers and a list of contacts*". Contacting other large organisations and asking them what they are using or conducting a market survey are other ways of identifying candidate products that the expert recommends.

E2 observes that "there is no reason why you cannot identify candidate products while you are eliciting high level requirements". However, with much larger products unlike e-mail packages, E2 suggests that there must be a *"pure requirements specification that does not have any bias towards any product"* before you can start identifying candidate products. Having a requirements specification that outlines what product capabilities are required before identifying existing candidates products avoids bias "in the way that you elicit the capabilities of the products". For e-mail packages, the best thing to do is to:

(i) go to the e-mail package conference and ask every participating vendor;
(ii) obtain all the vendor literature and on the basis of the customer requirements review most of the product but concentrate of 3-5 packages.

For larger COTS products such as ERPs the best thing to do is to visit vendors that have been shortlisted, talk to other people or organisations that are already using the product. From these collect both favourable and unfavourable experiences of product use. It is very important to involve the stakeholders very early on in the process and at every stage of the process.

Situation 3: insufficient product and supplier information – *E1 provided equivalent (C) but narrow matching advice (E) and exact matching techniques (A). E2 provided equivalent(C) but course match for both advice and techniques (D).*

E1: The product information depends on the results of searching the internet. This expert suggests that the ideal way to obtain product and supplier information that cost less is to "down load product information from the internet". However, the expert observes that "accessibility to some products may be very difficult". In this case the expert suggests sending "a request for information to the supplier or actually visiting them". "However, the key issue is to try to get a blanket picture of what capabilities are available" says the expert. Another way of obtaining product information that is suggested by the expert is if during requirements acquisition process the business events that the COTS product has to support have been identified, "you can then make a questionnaire based the business events and their responses that the customers want satisfied and send it to suppliers".

E2 suggests that from the identified products "you need to shortlist them to a manageable number of 4 or 5 products". To do this, "you need to get products marketing or vendor literature and then identify features of the products that will be definitely excluded". Examples of features of products that can be used to exclude them are like "if the customer works on NT and the product only works on workstation". In this way, "products can be very quickly eliminated using supplier marketing literature". Once you have short-listed them to a manageable list, you can do a more detailed analysis, matching capabilities of each product against each of the customer requirements. However, you need to be "careful when short listing products using marketing information, because this might result in excluding ideal products unless there is an adequate representation of the user requirements". The expert further observes that "if there is a sensible representation of customer core requirements you will find that a number of products will not actually meet the requirements. If they all meet them you then need to weight them on how well each of them do meet the requirements and take the top high scoring ones. However, this can be quiet a very subjective process".

Situation 4: no discriminating requirements – *both the advice and techniques provided by both experts did not match that provided by PORE (B).*

E1 recommends using 'fit criteria' to discriminate between products by asking each product to satisfy that fit criteria. For example, if the system has a requirement to send

messages, you need to know what 'send messages' means in terms of the customer requirements but not what the product can do to send messages. Once you have defined what 'send messages' means in terms of customer requirements, you can then determine if the product sends messages according to the customer's requirement definition of sending messages. However, the expert did not suggest any specific techniques.

E2 observes that it's "highly unlikely that all products will meet the requirements in exactly the same way. Some products will meet the requirements better than others". The expert suggests "focusing on these differentials and to weight each product on how well they meet each requirement". In this case the expert suggests "prioritising the requirements in order to determine which are more important and which products are mapped to those requirements and how well they meet those requirements". The expert also recommends using other types of requirements other than functional ones to "differentiate between products". Such examples of requirements that the experts recommend using are maintenance requirements, cost of the product, or the supplier's capabilities and customer base. In this case its highly unlikely that you will get all products that meet all these requirements equally and therefore, the non-functional requirements differentiate most. The expert does not suggest any specific technique to use in this situation.

Situation 5: reduce the number of candidate products to a manageable list – *E1 provided advice and techniques that matched (A) that provided in PORE. However, the advice was too simplistic (E). E2 provided equivalent (C) but course matching information for advice and techniques (D).*

E1 recommends choosing about "10 key requirements and do a quick check on all candidate products and see if you can eliminate some of them based on these key requirements".

E2 recommends first weighting and prioritising all customer requirements, then matching each product to functional and non-functional requirements, and allocating compliance scores. After allocating compliance scores, "select the top 5 or 6 products

and record the reasons for making the selection decision and at what particular point you took that decision so that you can later back track to the decisions made earlier".

Situation 6: evaluate in detail shortlisted products – *both experts provided equivalent (C) but course information for both advice and techniques (D).*

E1 suggested that "the key thing is to determine if all of the shortlisted products satisfy the 10 key requirements in exactly the same way". The expert further recommends to use a simple ordering scheme with probably three stages like meets perfectly, meets adequately, does not meet at all. This scheme enables comparisons between the products and a basis for comparing them.

E2 recommended analysing the requirements in more detail and scoring products against those detailed requirements. The expert also suggested that ideally, you have to have "established a method on how you are going to do the selection of the candidate products before you start the selection process". "You need to establish in advance how you are going to shortlist products, what techniques are you going to use to shortlist products, how are you going to record decisions and how are you going to make the final selection", says the expert. Without doing this, the expert suggests that a lot of time can be spent "wondering on non-important tasks". Once the products have been shortlisted into a manageable number, the expert recommends to "have vendor demonstrations and get people to use the products if possible". The vendor demonstrations can be used as a filtration exercise. For a large COTS product, things like the compatibility of the vendor's culture to the customer buying the product and vendor's flexibility to accommodate requests could be very significant selection factors.

Situation 7: deciding and selecting the final preferred product – the advice and techniques provided by *E1 did not match with that provided in PORE (B). Expert 2 provided advice that is equivalent to PORE's (C) but suggested techniques that did not match that provided in PORE (B).*

E1 recommended to "re-involve the stakeholders and show them the list of final products". Once the stakeholders have been shown the list, you need tell them that "its

not a definitive list and show them the differences between the products". When they have seen the products, "you then obtain feedback on their satisfaction/dissatisfaction about different requirements". The expert also recommended "involving stakeholders which have political influence and to look at the degree of influence of each stakeholder". This will help you "decide which product you are going to give more priority".

E2 suggested that "if a quantitative selection judgement cannot be made, you then need to make a qualitative judgement such as how well do the customer organisation get on with the supplier or is the supplier going to be able to respond to request for customisation or can we work with this supplier". The expert also recommends identifying "the differentiation factors between the products on how well they meet the requirements". To this, the expert suggested considering "soft issues or getting a group of stakeholder representatives together, describe to them the capabilities of each product and have a brainstorming session with them to identify what soft issues they would use as criteria". Once this is done, "you then get a group consensus as to how the products should be weighted for final selection". In this way "you can come up with some factors that can be used to discriminate the products and make the decision based on that".

The results of the reported expert evaluation provide evidence for the usefulness of PORE guidance. The evidence suggests that PORE's advice is as good as that provided by the experts and in some cases it is even more directed than that of experts. Partial evidence was also found for the predicted use of the recommended techniques. Therefore support has been found for hypotheses H6, stated in the thesis objectives:

Hypothesis 6: PORE advice is at least as good as expert advice and in some cases is more fine grained and directed than the experts.

6.4 Summary and Chapter Conclusions.

In summary, this chapter has discussed the evaluation of major components of the PORE method through case studies and experts evaluation. First the guidance provided by the first two PORE templates was evaluated using three case studies carried out in three different organisations. Both case studies were to select and recommend a COTS product. The PORE templates were found to be usable and were used. The advice and guidance provided in the templates was found to be useful. Important additional improvements to the templates were suggested. The usefulness and effectiveness of the advice and guidance provided by PORE was evaluated through expert study. Seven representative situations (scenarios) that covered all parts of the PORE method were presented to two requirements engineering experts and were asked to say what they would do if they were in that situation. The information given by the experts was compared for fit with that given by PORE. The results show that there is significant match but that most advice and guidance given by experts was general and already included in that given by PORE. The expert study showed that the PORE method is effective and beneficial in selecting COTS products. The results of the case studies and expert study therefore provided support for the theory predictions.

Chapter 7

Evaluating the PORE method

This chapter summarises the thesis research and concludes with a discussion of possible future directions.

Chapter 7
Discussion and Conclusions

7.1 Summary

This chapter summarises the work reported in this thesis and presents the benefits of the approach taken in providing solutions to the requirements engineering problems for COTS-based systems development. The research proposed an iterative parallel process of requirements acquisition and product selection. It also proposed the use of process goals, models and situations as a means of providing the evaluation team with process guidance. This chapter also describes some limitations of the method developed in this thesis and proposes future work that is needed to improve it. Particularly, the proposed method needs to be improved to address issues such as multiple COTS selection, the buy vs. build decisions, selection of application service providers (ASP) or internet service providers (ISP), the involvement of product suppliers as key stakeholders in selection process, the evolution of both the COTS product and the system developed from the COTS products and the lease vs. rent vs. loan decision. The chapter concludes by identifying future research directions on requirements engineering for COTS-based development paradigm.

The deliverables of the research are:

- two empirical studies of packaged-based development that demonstrated the need for a coherent method for acquiring requirements and selecting software products. The studies identified many unsolved problems and a general lack of requirements engineering research for packaged-based systems development;

- an iterative process model of requirements acquisition and product selection to guide evaluation activities during product procurement. The model dynamically provides process advice based on goals to be achieved and techniques to be used in a context defined by the current state of the compliance relationship between the requirement model and the product model;

- an approach demonstrated by four case studies which identified the usability and repeatability of the method and usefulness and effectiveness of the advice produced by the method.

In chapter one, the aims and approaches of this thesis were outlined. The objective was to develop a method to help address the problems of requirements engineering for COTS-based systems development. To structure and drive the research, 6 hypotheses were identified. Chapter two summarised previous and current trends into two main research areas. Research on requirements engineering has focused on methods, techniques and tools for bespoke systems development but has ignored packaged-based systems development. On the other hand, research on packaged-based systems development has focused on the architecture, design and integration, but has ignored the requirements acquisition and product selection phases which must precede design and integration.

7.2 Testing the thesis hypotheses

The research was driven by 6 hypotheses which are described below.

7.2.1 Hypothesis H1

Hypothesis H1 stated that '*new problems arise in the requirements engineering related phases of COTS-Based Development that are not addressed in current requirements engineering research*'. This was investigated in chapter 3 through two studies of packaged-based development processes. The first was a study of three organisations which developed COTS-based systems. The study was carried out to gain more comprehensive knowledge about software procurement. The second was a substantive study of the selection of a COTS requirements management system. The study identified problems which arise during requirements acquisition for COTS software selection. The first study identified 29 major problems that were experienced by all 3 organisations at all stages of the procurements process. Of the 29 problems, only 8 were found to be partially addressed by current COTS-based development and requirements engineering methods described in chapter 2. The second study also

identified these problems and reported on all of the problems in detail. As such evidence was found to support for hypothesis H1.

7.2.2 Hypothesis H2

Hypothesis H2 stated that '*it is possible to design more effective methods which directly address current problems in requirements engineering for COTS-based development*'.

This was investigated in chapters 3 and 4. Chapter 3 describes a new method that exists at two levels. The first level is a simple template-based process. The second level is an iterative process of customer requirements acquisition and COTS software product selection discussed in chapter 4. The features of the method directly address problems that were identified and described in chapter 3. The method has four main components:

- an iterative process model;
- a goal-based process guidance;
- a multi-layered situated process guidance;
- 3 essential models for guiding the selection process.

The method solves problems identified in studies reported in chapter 3. The effectiveness of the method in addressing the identified problems and the evidence to support hypothesis H2 is demonstrated in chapter 6.

7.2.3 Testing hypotheses H3 – H6.

Chapter 6 reported tests of hypotheses H3-H6 through evaluations of the PORE method. Three case studies in 3 different organisations and one exercise with requirements engineering and COTS software experts were carried out to test the usability and effectiveness of the PORE method. Three studies of the use of PORE to select and recommend a COTS product were used to test hypotheses H3 – H5. The first two studies (organisation A and B) tested hypotheses H3 & H4:

Hypothesis H3 stated that *'the PORE method guidance can be applied in part or in whole to real-world software product selection tasks'.*

Hypothesis H4 stated that *'The PORE method can form an essential part of a successful product selection task'.*

Hypotheses H3 & H4 were tested through 2 studies in two different organisations to select off-the-shelf software products. The first was a study to purchase a new dealerboard system for an international bank. The second was to purchase an anti-virus security system for an international merchant. In both cases, the requirements engineering team applied guidance from PORE's template 1 to identify candidate COTS products and to acquire essential customer requirements. PORE's template 2 guidance was used to organise and conduct supplier-led product demonstrations and to recommend the preferred products. In both studies, PORE's guidance guided the team to a successful conclusion. Therefore, support for hypotheses H3 has been found and the hypothesis is accepted. Partial evidence to support hypothesis H4 was also found and therefore this hypothesis needs to be further tested. This evidence further strengthen support hypothesis H2 in that the PORE method which is designed from identified problems led to a successful product selection.

The third study (organisation C) tested hypothesis H5 which stated that *'PORE guidance is perceived to be useful and usable by people involved in the product selection task'.* This was tested through a study to select a document imaging and management system for a Lloyds of London insurance syndicate. During each product demonstration session PORE's template 2 provided the team with guidance and advice on how to score product-requirement compliance. At the end of each demonstration session, template 2's guidance was used to collate the product's scores. After all shortlisted products have been demonstrated, template 2 was used to rank the products. A preferred product was selected and used.

At the end of the process, the team distributed a questionnaire to 5 stakeholders who were involved in the process to elicit their comments and perceptions about the usefulness of PORE's process guidance. All 5 stakeholders indicated that the process guidance was useful. They all indicated that the method provided a useful way of

systematically filtering out non-compliant products. The stakeholders also indicated that they would use PORE again if there were to select another software product in the future. From this evidence it was possible to accept both hypotheses H4 and H5.

Hypothesis H6 stated that *'PORE's advice is at least as good as expert advice'*. This was tested through elicitation of knowledge from experts in requirements engineering and software package selection and an expert critiquing of the PORE method. Two requirements engineering consultants from two different organisations with 38.5 years of experience between them in requirements engineering and COTS product selection in a variety of application domains took part. The experts were presented with 7 focused requirement acquisition and product selection scenarios that were chosen to provide coverage of all parts of the method. The experts were asked to described what they would do in that situation and what techniques they would use. At the end of the study, the experts' results were compared with PORE's predicted advice and guidance. In 2 cases the expert advice matched exactly that is provided by PORE; in 3 cases the expert advice did not match that provided by PORE; in the remaining 9 cases, the advice provided by experts was either narrow or to general than that provided by PORE. Overall, the advice provided by PORE included that provided by the experts and in some cases it was better than that of experts. The evidence therefore supports the acceptance of hypothesis H6.

The remainder of this chapter discusses contributions of the thesis research and possible future work in PORE and future research directions.

7.3 Contributions to research on requirements engineering for COTS-based systems development

This research has improved and contributed to our understanding of the problems of requirements engineering for COTS-based systems development. The research highlighted existing requirements acquisition and product selection issues that are important in informing the development of methods and tools for COTS-based systems development.

The deliverables of this research are:

- A method, PORE, is proposed to fill the gap in requirements engineering methods and guidance for the CBD development process. The method supports an iterative process of requirements engineering and product evaluation/selection;
- A process model that identifies key decisions points that should be made by any CBD process and four generic processes for achieving each decision point and a sequence of achieving these decisions is developed;
- A software product model, a requirements model and a situation model that models compliance relationships between software product and customer requirements;
- Strategies for guiding the COTS-Based Development process using models, goals and situations from a range of disciplines;
- A concept demonstrator prototype process advisor tool for guiding and advising requirements engineers;
- Empirical evidence to support the usability and usefulness of the PORE method advice and guidance.

This thesis has provided theoretical work that helps to undertake the process of acquiring requirements for COTS-Based systems development. The importance of requirements for COTS software selection has been widely recognised, for example by Carney (1999) and Davies (1999). In traditional systems development paradigm, requirements are elicited from users and analysed and then the system is developed from these requirements. However, developing systems from off-the-shelf packages requires a new approach and culture (Lattaze 1997). Requirements engineers, developers and user organisations will need to be trained on how to acquire the requirements. The iterative approach described in this thesis recognises that the process of requirements acquisition and system development are intertwined. As such, the stakeholders are interactively involved in both acquiring requirements and developing the system. The advantages of this approach has been recognised, for example, by Polydys (1999) and Swannson (1999).

7.4 Discussion

This thesis addresses requirements engineering for COTS-based systems development. The thesis focuses on the processes of the requirements acquisition and COTS product selection. However, the thesis does not address issues related to multiple COTS selection, the buy vs. build argument, Application Service Providers (ASP), product suppliers as suppliers as key stakeholders and the evolution of the COTS products and COTS developed system. Although, these issues are outside the scope of the thesis, the PORE method and the techniques described in this thesis can contribute in providing solutions to these problems. The following sections discuss how PORE needs to be extended to contribute to these problems.

7.4.1 Multiple COTS Selection

In a multiple COTS system, many disparate products from different often competing suppliers are integrated, glued and combined to provide system functionality that is unavailable from any single suppler (Obendorf 1999). Evaluating and selecting COTS software products for use in a multiple COTS system is far more complex and difficult than evaluating single COTS software (Obendorf 1999). The selection decisions in a multiple COTS system depend on the selection of other products resulting in dependencies among selection decisions. Similarly, these products have independent lives and evolve at different speeds and this has to be taken into account during requirements acquisition and in the final selection decision. The key point in multiple COTS selection is that the evaluation activity must be focused on integration as a cohesive unit (Vigder et al. 1996). This has important implications on the requirements engineering activities that are not addressed by current practices including research described in this thesis. The requirements acquisition process must be able to deal with requirements for selecting individual products and at the same time deal with the requirements for the overall system to be produced from assembling individual products. There is a need to identify requirements that deal with the system as a whole, requirements that deal with individual products and those that are common to both. To do this, two kinds of product evaluation and requirements acquisition strategies are required (Carney 1999).

In the first strategy, product alternatives to implement different parts of the system are evaluated using essential customer requirements (e.g. functional, non-functional and architecture requirements). In the second strategy, alternative ensembles are evaluated using architecture requirements. For this, each product is evaluated in relation to other products. In both strategies, the decision to select a product depends on the selection of other products therefore resulting dependencies among selection decisions, products and customer requirements. Also, the desired attributes of the system are realised through the combination of the desired attributes of the individual COTS products therefore creating more dependencies among product attributes. Another characteristic feature of a multiple COTS development is that the usability of the individual products might not be important but the emphasis is on the usability of the resulting system developed from the selected products. The usability requirements for each candidate product are less important in a multiple COTS selection than in a single COTS selection.

As mentioned in section 1.3, the present version of PORE does not address multiple COTS selection. In multiple COTS selection, two sets of requirements types need to be identified - requirements that focus on evaluating alternative products and customer requirements that focus on evaluating the system that is to be composed from the selected alternative products. The present version of PORE focuses on identifying customer requirements for selecting individual stand-alone products. The future version PORE needs to be extended to address the multiple COTS selection. The method needs to provide the requirements engineering team with advice and guidance to identify the desired attributes (i.e. functional, non-functional and architectural attributes) of the candidate products on one hand and the desired attributes of the overall system on the other hand during the evaluation and selection process. Also, new techniques, evaluation and selection strategies and new process advice and guidance need to be identified and added to the future version PORE.

7.4.2 Supplier View

In COTS-based development process, selecting a COTS product usually means selecting the product's supplier as well. Therefore, in order to make a successful COTS product selection, it is necessary for customer organisations to consider the suppliers as stakeholders in the selection process. COTS product selection is just as much a decision about business relationships as is about system functionality. Supplier requirements or relationships are just as important as functional requirements. In a system comprising of multiple COTS products that must be integrated with each other, managing supplier relationships is just as important. However, this is not addressed in the current COTS development methods including the current version of PORE.

In a large-scale multiple COTS system, suppliers play a far more active role than just as providers of commodity items as products are viewed. Therefore, the way in which contractual or supplier relationships are handled is a key component in ensuring that the solution/product being procured will meet the business/system requirement with minimal risks. In the production of a multi-COTS system, the interest is not only in integrating products but also in integrating suppliers as well. However, the method described in this thesis does not adequately address the role of the supplier in product selection and does not recognise suppliers as key stakeholders in the selection process. The method ignores the influence of supplier issues which are far more than can be simply accommodated solely by criteria involving factors such as supplier size or financial health as in the present version of PORE. The PORE method needs to be extended to appreciate the dualism of product and its supplier. The method needs to appreciate that the decision to select a product should take into account not only its integration with other products but also the integration of the suppliers of those products as well. The current version of PORE solely focuses on the views of the customer and therefore needs to be extended to include suppliers as key stakeholder as well.

7.4.3 The Buy vs. Build Decision

Custom-building systems ensures organisations that the system is exactly as needed even though the proposition to build is always potentially expensive and time-consuming. On the other hand, buying off-the-shelf software is often cheaper but sometimes may require that organisations have to modify their processes to conform to the software. Which option is right for any organisation depends on the organisation's technological expertise, financial status, the size of the organisation and the uniqueness of the business. Therefore, when deciding which course of action to take organisations need to carefully consider the costs and risks of both paths and choose the one that provide more value over the long run. However, many currently existing methods including PORE do not address the buy vs. build argument. Most COTS-based development methods including PORE assume that the decision to buy has already been made. The PORE method exclusively focuses on the buy vs. buy decisions and ignores the fact that organisations may want sometimes want something in the middle where some parts of the system are custom build and some parts are bought off-the-shelf. Therefore, future version of PORE needs to be extended to include trade-off analysis between buying an off-the-shelf solution and custom-building it.

7.4.4 Application Service Providers

Application Service Providers (ASPs) rent applications to customers who access them via the internet. The growth of the internet, intranets and extranets has revived the concept of renting, leasing or loaning software. The rentable software concept is growing and is giving organisations fast access to a variety of applications without requiring a major upfront investment. Rentable software is aimed at organisations that do not have the resources to install a complete system. By eliminating upfront costs, it is possible for organisations to use applications without physically obtaining a copy and having to bear the cost. Renting enables organisations to have access to specialised applications that would otherwise be too expensive to implement. Renting software is great for certain situations such as specialised applications like groupware applications, project management applications, document management and change control applications. One of the benefits of renting software is that organisations don't

have to buy, store, install or maintain the software (Kay 2000). This also eliminates the 'antiquated model' of software licensing. Software licensing is wasteful and its time-consuming to keep track of all upgrades. Most rented software share a common sales pitch: save time, save money and save face by avoiding long-term commitment to a single platform. A variety of models payment have been proposed. These include pay-per-use (e.g. as in telephone services), flat rate on a monthly basis (e.g. as in renting a house) or rent and pay per transaction. Although the ASP business model is gaining popularity, currently, there are no methods that help customers to evaluate and choose preferred service provider. The PORE method does not directly address the problem of choosing an ASP. However some of the lessons learned and techniques developed in this thesis can help organisations and individuals evaluate and select a suitable application service provider. Future versions of PORE need to be extended to directly address the problems of associated with choosing an ASP.

7.4.5 Requirements for Product Evolution

Three main factors generally influence products or systems evolution:

- when its specification changes (i.e. a new version of the product);
- when customer's needs change (e.g. new requirements or regulations);
- when the product or system's operation environment changes.

In a COTS-based system, product evolution can result in far more serious consequences than in a bespoke system. The problem for COTS-based development is that there are two independently evolutionary cycles that take place simultaneously – the evolution of the individual products and the evolution of the system itself. This is even far more complex in a multiple COTS system in that different products from different suppliers will be evolving independently of each other. However, although these problems represent potential risks to customers, they are not generally addressed by current methods when selecting COTS products. Most methods including PORE, do not focus beyond the selection of the preferred product and address the problems caused by the both the evolution of the customer's system and the supplier's product. Another problem is that suppliers may find some incentives to adapt their products to meet requirements for key larger customers at the expense of smaller organisations

thereby leaving them with orphaned products. Therefore, when selecting COTS products, it essential that due care must be paid to product's and system's evolutionary requirements. Although PORE recognises the importance of contractual requirements, these are not adequately addressed in the current version. The method needs to be further extended to address these problems.

The following section describes future planned work that aim to improve PORE and to address the problems identified in previous chapters and in sections 7.4.1 – 7.4.5.

7.5 Future work to improve PORE

The method reported in this thesis aims to address problems in requirements engineering for package-based development. Since the technology constantly changes, up-to-date research is required both on a short and long scale. The following sections describes the envisioned short-scale and long-scale future work.

Although the thesis contributed useful research in requirements engineering for COTS based development, the current version of the PORE method has some weaknesses that were identified in the evaluation studies and some limitations that were discussed in section 7.4:

- PORE takes too long use. PORE requires alternative pathways to allow tailoring for faster use. The present version of PORE assumes that all parts of the method should be performed in each case. However so far, the method has been applied on limited scale, in small studies. A large, industrial scale use of PORE is needed in order to be able identify different routes to enable it to be tailored.

- the PORE templates will be changed in light of results from the evaluation studies. One solution would be to make the templates advice guide users on the different pathways that they can take based on the size and complexity of the COTS system being developed. At their present form, the templates provide similar advice for all types of evaluations.

- the PORE method and the prototype tool can be too complex. A comprehensive tool is needed to support the PORE method.

- the PORE method needs to address multiple COTS selection. The present version focuses on selecting a single COTS software product. The method also needs to address the buy vs. build argument. The current version focuses on the buy vs. buy decision. Customers need to be given an option to choose whether to buy the solution off-the-shelf or to custom-build or to do both. At the moment PORE does not give that choice. PORE recommends that stakeholder representatives should be present during product demonstration. However, the present version does not include product suppliers as key stakeholders. The method needs to include suppliers as stakeholders in process as well. Although the lessons learned and techniques developed in this thesis can help in the selection of application service providers, the PORE method needs to be extended in order to understand and adequately address the problems associated with choosing an ASP. PORE does not address the problems associated with the evolution of both the COTS products and the system itself. At present, the method focuses on selecting the preferred product but does not deal with the development of the system or the update of the COTS products after it has been procured. The method needs to be extended to include product updates and evolution of the system as part of its process.

7.6 Future research directions for requirements engineering for COTS-based systems development paradigm

Since this thesis is one of the first to explore requirements engineering for COTS-based systems development, there are 3 possible future research directions. Each is described in turn.

7.6.1 COTS software simulation environments

A more complete modelling and simulation environment is needed in order to understand many competing, critical issues that arise during requirements engineering for CBD. This environment model will take as input, customer requirements (both

functional and non-functional), current legacy systems, software products and the glueware software to explore the selection and integration of COTS products into their environment. One critical success factor will be a plug-and-play environment in which models of different COTS software products can be plugged in to explore their consequences. The simulated environment would also allow to evaluate and predict reliability of COTS based systems and to build reliability models of these systems and could be incorporated into the PORE method process. The concept of this simulated environment is depicted in figure 7.1.

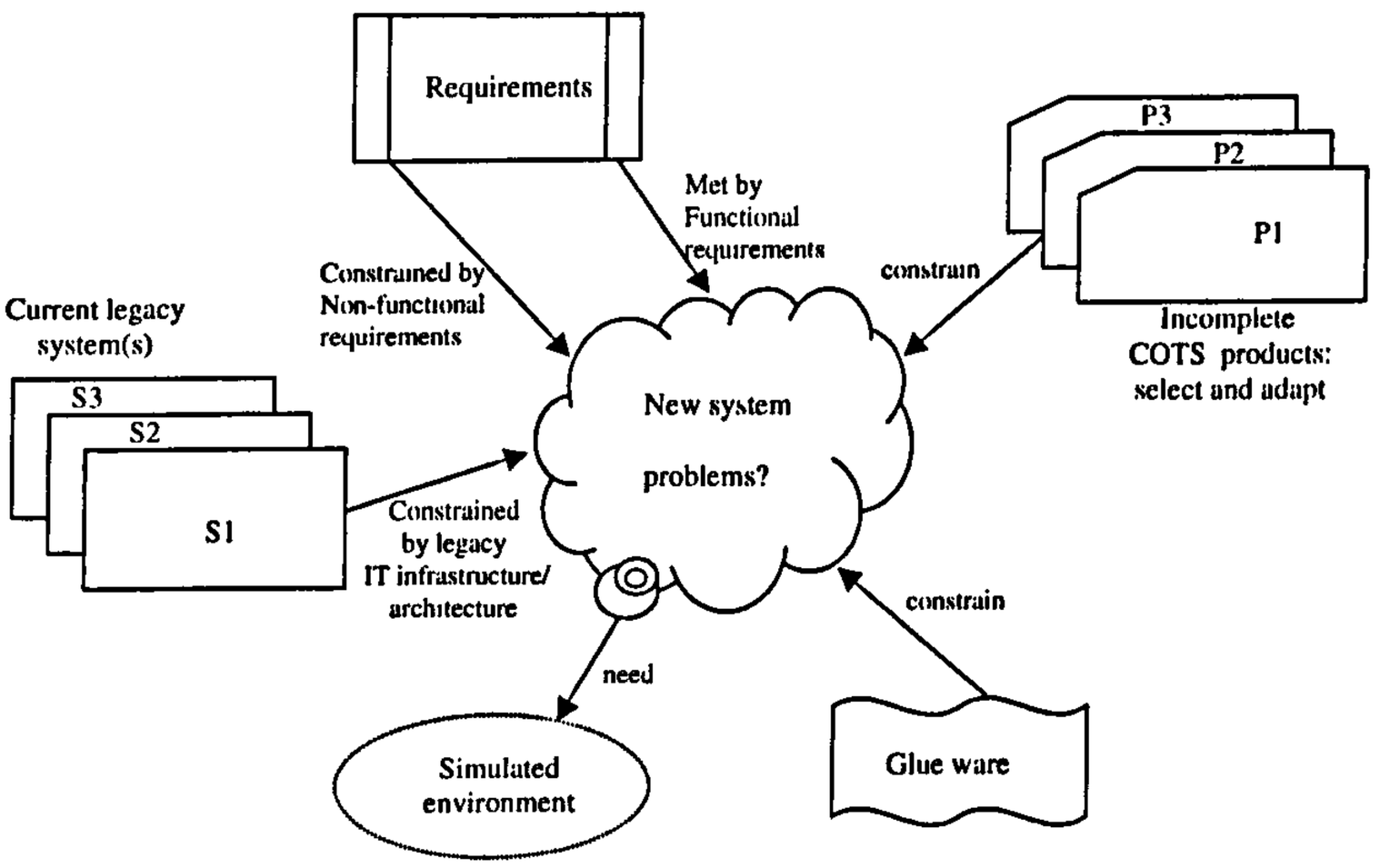


Figure 7.1. A modelling and simulation environment that allows plug-and-play of models of COTS software packages and to model reliability requirements of COTS-based systems.

7.6.2 A Shared Knowledge Development Process

One of the lessons learned from the study carried out in organisation C described in chapter 6 was that stakeholders indicated that suppliers should be involved in COTS software package selection decisions. Indeed, since the COTS software products are developed by suppliers who have control over them, it is essential to involve suppliers in the process. Therefore, one 'vision' is of a process whereby shared knowledge of products, development skills and experiences, new techniques and methods is made available to all participants in the process. This shared knowledge will result in a supply chain of components/products, skills and experiences and personnel.

Organisations will also have access to each other's development infrastructure along the supply chain and share business strategies and objectives, expertise, ideas, risks and information. Joint technology development programmes will be possible through shared knowledge and close relationships. This vision of a shared knowledge process is depicted in Figure 7.2.

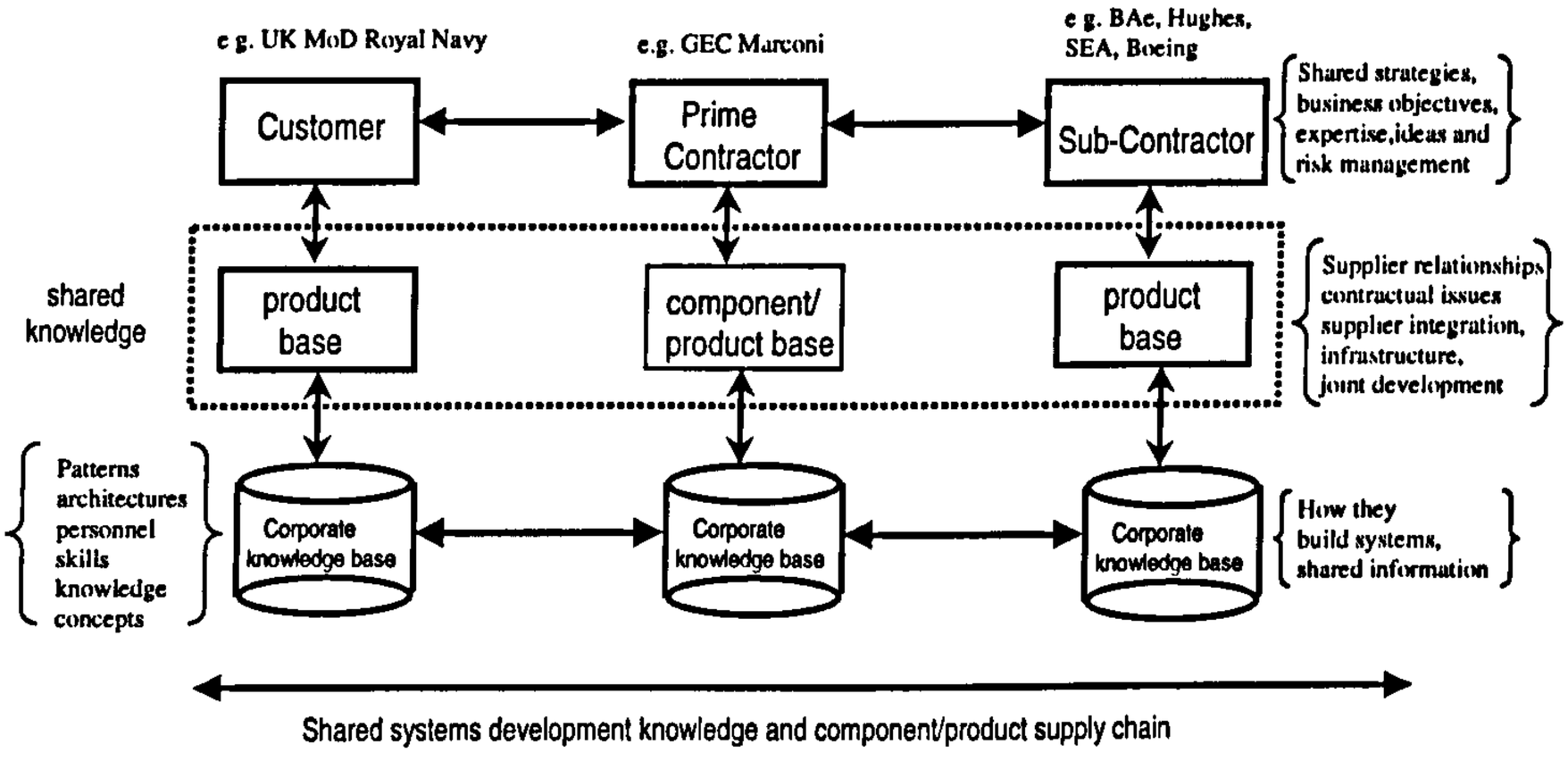


Figure 7.2: A vision of shared knowledge as the cornerstone for the future success of the CBD paradigm.

7.6.3 The 'soft' Issues: Training and Education

The techniques and knowledge from different disciplines required in the CBD process for it to be a success will mean that it will be impossible for any individual to possess all of the necessary skills and knowledge. As a result, the development team of the future will be composed of team members from many backgrounds to form 'smart teams'. The project management and development skills that are required for the CBD process are significantly different from those required for traditional systems development. Organisations themselves will probably have to change the way they do their business. For example, selecting a product to be included in the integrated systems largely results in the selection of the product supplier (Carney 1999, Wallnau et al. 1998). Therefore, in a COTS-intensive system, the integration of the different products results in the integration of product suppliers (Oberndorf el al. 1997b, Vigdar et al 1997). As such, the development organisation will need to manage not only its relationships with the individual suppliers but also the relationships between the

integrated suppliers (Allen 1998). As a result, this thesis identifies personal or 'soft' issues as another major research area. A different set of skills will be required and therefore CBD principles need to be incorporated into the training and education of systems developers and requirements engineers of the future, be it in universities, colleges or organisation training programmes.

So to conclude, this research work provides important early contributions to research on requirements engineering for COTS-based systems development. It is hoped that the work reported in this thesis will be a useful basis for future work and will encourage further research in this area.

Primary References

ANDERSON E. E. (1989). A Heuristic for Software Evaluation and Selection, Software Practice and Experience, vol 19(8), August 1989

ANTON I. A. and POTTS C. (1998). The Use of Goals to Surface Requirements for Evolving Systems, to appear in Proceedings IEEE International Conference on Software Engineering, IEEE Computer Society.

ANTON I. A. (1997). Goal Identification and Refinement in the Specification of Software Based Information Systems, PhD Thesis, Georgia Institute of Technology, June 1997.

BOEHM, B.W. (1988). A Spiral Model of Software Development and Enhancement' Computer, May 1988

BROOKS F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering, IEEE Computer, April 1987, pp10-19

BROWN W. A and WALLNAU C. K. (1998). An Examination of the Current State of CBSE: A Report on the ICSE 98 Workshop on Component-Based Software Engineering, Japan, 1998.

BROWN W. A. (1998). From Component Infrastructure to Component-Based Development, Proceedings CBSE98 for ICSE98, Japan 1998.

BROWN A.W. and SHORT K. (1997). On Components and Objects: The Foundations of Component-Based Development, Proceedings 5th International Symposium on Assessment of Software Tools and Technologies, IEEE Computer Society, 112-121.

BROWN A.W., CARNEY D.J. and McFALLS M.D. (1995). Proceedings SEI/MCC Symposium on Use of COTS in Systems Integration', Technical Report CMU/SEI-95-SR-007, Software Engineering Institute, Carnegie Mellon University, June 1995.

BUCKINGHAM S. S. and HAMMOND N. (1994). Argumentation-Based Design Rationale: What Use at What Cost?', International Journal of Human-Computer Studies 40(4), 603-652.

CARNEY D. (1999). COTS Product Evaluation and System Design, The COTS Spot, SEI Interactive, Volume 2, Issue 1, March 1999.

CARNEY D. (1998). Quotations from Chairman David: a little Red Book of Truths to Enlighten and Guide on the Long March Toward the COTS Revolution; July 1 1998.

COSTELLO J and LIU D. B. (1995). Journal of Systems Software, 29: 39-63

DARDANNE A, van LAMSWEERDE A and FICKAS S. (1993). Goal-Directed Requirements Acquisition, Science of Computer Programming 20, 3-50

DARIMONT R. and van LAMSWEERDE A. (1996). Formal Refinement Patterns for Goal-Driven Requirements Elaboration, Proceedings of 4th ACM Symposium on the Foundations of Software Engineering (FSE4), San Francisco, Oct. 1996, pp 179-190.

DAVENPORT H. T. (1996). Holistic Management of Megapackage Change: The Case of SAP, Proceedings 2nd Americas Conference on Information Systems, Phoenix, Arizona, August 16 – 18, 1996

DAVIES A. M. (1993). Software Requirements: Object, Functions, State, Prentice-Hall

DATAQUEST (1997) http://gartner6.gartnerweb.com/dq/static/dq.html

DEAN C. J and VIGDER R. M. (1997). System Implementation Using Commercial Off-The-Shelf Software, Proceedings of the 1997 Software Technology Conference (STC 97), Salt Lake City, Utah, May 1997

DEAN C. J. (1999). Timing the Testing of COTS Software Products, Testing Distributed Component-Based Systems, 1ST International ICSE Workshop, LA, California, USA, 17 May 1999

DOBSON J. and STRENS R. (1994). Organisational Requirements Definition for IT Systems, Proceedings of 1st International Conference On Requirements Engineering, IEEE Computer Society Press, 158-165.

DOWELL J and FINKELSTEIN A. C. W. (1996). A Comedy of Errors: The London Ambulance Services Case Study', Proceedings 8th International Workshop on Software Specification and Design, IEEE, Computer Society Press, 141 –145

EVOLVING ENTERPRISE. (1998). Volume 1, Number 1, Spring 1998.

FENTON N. (1994). Out Ranking Method for Multi-Criteria Decision Aid: with emphasis on its role in systems dependability assessment, Centre for Software Reliability, City University, London, UK.

FINKELSTEIN A. C. W, SPANOUDAKIS G and RYAN M. (1997). Software Package Requirements and Procurement, Proceeding 8th International Workshop on Software Specification and Design, IEEE Computer Society Press, Los Alamitos, Califonia, 1996, pp 141-145

FOX G, LANTNER K and MARCOM S. (1998). A Software Development Process for COTS-Based Information System Infrastructure (part 1), Cross Talk, March 1998

FORRESTER RESEARCH (1997) http://www.forrester.com/

FORTUNE MAGAZINE (1997) http://www.pathfinder.com/fortune/

FOX G., MARCOM S. and LANTNER K. (1997). A Software Development Process for COTS-based Information System Infrastructure, Proceedings of the 5th

International Symposium on Assessment of Software Tools and Technologies (SAST'97), pp133-143

FRAIR L. (1995). Student Peer Evaluations Using the AHP Method, Foundation Coalition, Department of Industrial Engineering, University of Alabama Tuscaloosa, 1995

GARLAN D., ALLEN R. and OCKERBLOOM X. (1995). Architectural Mismatch or Why it's hard to build systems out of existing parts, Proceedings 17th International Conference on Software Engineering, IEEE Computer Society Press, 1995.

GAUSE D.C and WEINBURG G. M. (1989). Exploring Requirements, Quality Before Design, Dorset House

GRAHAM I. (1996). Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis, Object-Oriented Systems 3, 123-142.

GROSZ G., ROLLAND C. and MAIDEN N.A.M., Modeling Domain Knowledge for Requirements Engineering: A Process View, Proceedings IFIP Joint 8.1/13.2 Conference on Domain Knowledge for Interactive System Design, Chapman Hall, 102-116.

GRUNDY J. (1999). Aspect-Oriented Requirements Engineering for Component-Based Software Systems, Proceedings of the 4th International Symposium on Requirements Engineering (RE99), IEEE Computer Society,

JOHNSON W L., FEATHER M S., and HARRIS D R. (1992). Representation and Presentation of Requirements Knowledge, IEEE Transactions on Software Engineering 18(10) 853-869

KARLSSON J. and RYAN K. (1997). A Cost-Value Approach for Prioritising Requirements, IEEE Software 14(5), 67-74.

KEMP A. (1994). Software Procurement and Superconcurrent Engineering, IEE Computing and Control Engineering Journal, December 1994,

KITCHENHAM B and JONES L. (1997). Evaluating Software Engineering Methods and Tools: Part 5, The Influence Of Human Factors, Software Engineering Notes 22(1).

KITCHENHAM B. (1996). DESMET: A method for evaluating Software Engineering methods and tools, Technical Report TR96-09, Department of Computer Science, University of Keele, August 1996

KONTIO J. (1995). OTSO: A Sysmatic Process for Reusable Software Component Selection, Technical Report Number CS-TR-3478, 1995, University of Maryland, USA.

KONTIO J. (1996). A Case Study in Applying a Systematic Method for COTS Selection, Proceedings of the 18th International Conference on Software Engineering, IEEE Computer Society Press.

KRUCHTEN P. (1998). Modelling Component Systems with the Unified Modelling Language, Proceedings of the CBSE98 of the ICSE98 Conference, Japan, 1998

MAcCRIMMON R. K. (1972). Proceedings of Multiple Criteria Decision Making, University of South Carolina, October 26-27, 1972

MAIDEN N.A.M. and RUGG G. (1996a). ACRE: Selecting Methods For Requirements Acquisition, Software Engineering Journal 11(3), 183-192.

MAIDEN N.A.M. and RUGG G. (1996b). ACRE: Selecting Methods for Requirements Acquisition, Software Engineering Journal 11(5), 281-292.

MAIDEN N.A.M., MINOCHA S., MANNING K. and RYAN M. (1998). CREWS-SAVRE: Scenarios for Acquiring and Validating Requirements, Proceedings 4th International Conference on Requirements Engineering (ICRE98), IEEE Computer Society Press.

MAZZA C., FAIRCLOUGH J., MELTON B., De PABLO D., SCHEFFER A. and STEVENS R. (1994). Software Engineering Standards, Prentice Hall.

McGREW G and VIEGA J. (1999). Why COTS Software Increases Security Risks, Testing Distributed Component-Based Systems, 1ST International ICSE Workshop, LA, Califonia, USA, 17 May 1999

MORAN T.P. and CARROLL J.M. (1996). Design Rationale: Concepts, Techniques and Use, Hillsdale NJ: Lawrence Erlbaum Associates.

MORISIO M and TSOUKIAS A. (1997). A Methodology for the Evaluation and Selection of Software Products, Dipartimento di Automatica e Informatica, Politicnico di Torino, Italy.

NING Q. J. (1999). A component Model Proposal, Proceedings of the CBSE99 Workshop of the ICSE 99 Conference, Los Angels May 1999.

OBERNDORF P. (1997). Facilitating COTS-Based Systems: COTS and Open Systems, in Proceedings of the 5th International Symposium on Assessment of Software Tools (SAST'97), IEEE Computer Society, Los Alamitos, California, June 1997

PAUL J. R. (1993). Why Information Systems Disappoint. In the Proceedings of the XV International Conference on Information Technology Interfaces (V. Ceric and V.H. Dobric, Eds.) (15-18 June, Pula, Croatia). University of Zagreb Computer Centre, Zagreb, Croatia. pp. 19-24.

PLACE P. (1999). Requirements Engineering Roundtable, SEI Interactive, Volume 2, Issue 1, March 1999

PLIHON V. and ROLLAND C. (1995). Modelling Ways of Working, Proceedings 7th International Conference on Advanced Information Systems Engineering, CAiSE'95, Springer Verlag

POTTS C. (1995). Invented Requirements and Imagined Customers: Requirements Engineering for Off-The-Shelf Products, Proceedings 2nd International Conference on Requirements Engineering, IEEE Computer Society Press, 128-130.

POWELL A., VICKERS A., LAM W., WILLIAMS E. and COOKE B. (1997). Evaluating Tools to Support Component Based Software Engineering, Proceedings 5th International Symposium on Assessment of Software Tools and Technologies, IEEE Computer Society Press, 80-90.

RAY M. (1996). Dependency Diagrams, Cross Roads, The ACM Student Magazine, vol 2(3), February 1996.

ROBERTSON S and ROBERTSON J. (1999). Mastering the Requirements Process, Addison-Wesley

ROBERTSON S. (1998). Volere: Requirements Specification Templates, Edition 6, Atlantic Systems Guild, http://www.atlsysguild.com/GuildSite/Robs/Template.html,

ROLLAND C. and GROSZ G. (1994). A General Framework for Describing the Requirements Engineering Process, IEEE Conference on Systems, Man and Cybernetics, CSMC94, IEEE Computer Society Press.

ROYCE W. W. (1987). Managing the Development of Large Software Systems: Concepts and Techniques, Proceedings of ICSE9, IEEE Computer Society Press, 1987

RUGG G. and McGEORGE P. (1995). Laddering, Expert Systems 12(4), 339-346.

SAATY L.T. (1990). The Analytic Hierarchy Process (AHP): How to make a decision, European Journal of Operational Research, 48(1990), 9-26

SAATY L.T. (1990). The Analytic Hierarchy Process, New York, McGraw-Hill.

SCHMIDT R and ASSMANN U. (1998). Concepts for Developing Component-Based Systems, Proceedings of the CBSE98 of the ICSE98 Conference, Japan, 1998

SEACORD C. R, HISSAM A. S and WALLNAU. C. K. (1998). AGORA: A Search Engine for Software Components, Software Engineering Institute, Carnagie Mellon University, USA.

SOFTWARE ENGINEERING INSTITUTE (SEI). (1998). Architecture Tradeoff Analysis (ATA) and Software Architecture Analysis Method (SAAM), Software Engineering Institute, Carnegie Mellon University, USA, http://www.sei.cmu.edu/ata/ata_init.html

SHAW M. (1996). Truth vs Knowledge: The Difference Between What a Component Does and What We Know It Does, Proceedings 8th International Workshop on Software Specification and Design, IEEE Computer Society Press, March 1996

SJAAK B. (1999). Requirements Engineering for ERP: Requirements Management for the Development of Packaged Software, Proceedings of the 4th International Symposium on Requirements Engineering (RE99), IEEE Computer Society, June 1999

SOMMERVILLE I. (1992). Software Engineering, Addison-Wesley 4th Edition

SUCHMAN L. (1987). Plans and Situated Actions: The Problems of Human-Machine Communication, Cambridge University Press.

TEPANDI J. (1995). Quality Requirements and Decision-Making in Software Procurement, Tallinn Technical University, Estonia

THAYER L. R and DORFMAN L. (1997). Software Requirements Engineering, 2nd Edition, 1997.

THOMAS B. (1999). Meeting the Challenges of Requirements Engineering, Spotlight, SEI Interactive, Volume 2, Issue 1, March 1999

TRAN V. and LIU D. (1997). A Procurement-centric Model for Engineering Component-based Software Systems, Proceedings of the 5th International Symposium on Assessment of Software Tools and Technologies (SAST'97), pp70-80

VIGDAR R. M, GENTLEMAN W. M and DEAN C. J. (1996). COTS Software Integration: State of the Art, National Research Council, Canada, NCR Report Number 39198, 1996.

VIGDER R. M. and DEAN C. J. (1997). Architectural Approach to Building Systems from COTS Software, Proceedings of the 1997 Center for Advanced Studies Conference (CASCON 97), Toronto, Ontario, 10-13 November 1997

VOAS J. (1998). Maintaining Component-Based Systems, IEEE Software, vol. 15, no 14, pp 22-27, July/August 1998 Issue

WALLANAU C. K, CARNEY D and POLLAK B. (1998). How COTS Software Affects the Design of COTS intensive System, SEI June 1998 Report.

WATERS N. (1995). Systems Architecture and COTS Integration, Proceedings of the SEI/MCC Symposium on the Use of COTS in Systems Integration, SEI Special Report CMU/SEI-95-SR-007, June 1995

WATTS S. H. (1989). Managing the Software Process, Addison-Welsey.

ZAVE P. (1995). Classification of Research Efforts in Requirements Engineering, IEEE Symposium in Requirements Engineering

Secondary References

ACHOR C.B. and ROLLAND C. (1997). Introducing Genericity and Modularity of Textual Scenario Interpretation in the Context of Requirements Engineering, CREWS Technical Report, Centre de Recherche en Informatique, Universite de Paris 1, Paris, France.

ALLEN P. (1998). Component-based Architecture: A Call for Pragmatism, in the Proceedings of the CBISE98 of the CAiSE98, Pisa, Italy, June 1998.

BARON J. (1992). Thinking and Deciding, Cambridge University Press

BEUS-DUKIC L and WELLINGS A. (1998). The Requirements for COTS Software Component: A Case Study, Conference on European Industrial Requirements Engineerinhg, CEIRE'98, 19-20 October 1998, London, UK

CARNEGIE MELLON UNIVERSITY, CBS Overview, Software Engineering Institute, <URL: http://www.sei.cmu.edu/cbs/overview.html

CARNEY D. (1997). Assembling Large Systems from COTS Components: Opportunities, Cautions and Complexities, Software Engineering Institute, Carnegie Mellon University, June 20 1997.

CLEMENTS C. P. (1995). From Subroutines to Subsystems: COTS-Based Systems, American Programmer, v8 no.11, Cutter Information Corp, November 1995

CLEMENTS P., KAZMAN R. and ABOWD G. (1995). Predicting Software Quality by Architecture-Level Evaluation, Proceedings 5th International Conference on Software Quality, Austin 1995, 485-498.

CLEMENTS P.C. (1996). Coming Attractions in Software Architecture, Technical Report CMU/SEI-96-TR-008, Software Engineering Institute, Carnegie Mellon University, January 1996.

COLLINS D. C. (1972). Applications of Multiple Criteria Evaluation, in Proceedings of Multiple Criteria Decision Making, University of South Carolina, October 26-27, 1972

EASTERBROOK S. (1993). Domain Modeling With Hierarchies of Alternative Viewpoints, Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 65-72

FOWLER P. (1998). Requirements engineering and industrial uptake, Proceedings 4th International Conference on Requirements Engineering (ICRE98), IEEE Computer Society Press

FRANKEL Y. and ORR G. (1996). Commercial Off The Shelf (COTS) Software Evaluation: Report for Maintaining and Upgrading the Ground System (MUGSy), Technical Report, Goddard Space Flight Center, Greenbelt, Maryland, March 1996

GOTEL O C.Z. and FINKELSTEIN A. C. W. (1994). An Analysis of the Requirements Traceability Problem. Proceedings of the International Conference on Requirements Engineering (ICRE'94), pp 94-101, Colorado Springs, Colorado, USA, April 1994

GREEN G. R.T. and BENYON D. (1996). The Skull Beneath the Skin: Entity-Relationship Models of Information Artefacts, International Journal of Human-Computer Studies 44(6), 801-828.

GREENSPAN S., BORGIDA A. and MYLOPOULOS J. (1986). A Requirements Modeling Language and its Logic', Information Systems 11(1), 9-23.

HALL R.P. (1989). Computational Approaches to Analogical Reasoning: A Comparative Analysis, Artificial Intelligence 39, 39-120.

KOLODNER J.L. (1993). Case-Based Reasoning, Morgan-Kauffman.

LOUCOPULOS P. and KAVAKLI E. (1994). Enterprising Modelling and the Teleological Approach to Requirements Engineering, Department of Computing, UMIST, Manchester, UK

MAIDEN N.A.M. and SUTCLIFFE A.G. (1996). Analogical Retrieval in Reuse-Oriented Requirements Engineering, Software Engineering Journal 11(5), 281-292.

MAIDEN N.A.M., SPANOUDAKIS G. and NISSEN H.W. (1996). Multi-Perspective Requirements Engineering Within NATURE, Requirements Engineering Journal 1(3), 157-169.

MORRIS P., MASERA M. and WILIKENS M. (1995). Industrial Workshop on Requirements Engineering: a Report on the Results Obtained, Technical Report TP210, Ispra Italy.

MULLER P.C., de POORTER R., de JONG J. and van ENGELEN J.M.L. (1996). Using the Internet as a Communication Infrastructure for Lead User Involvement in the New Product Development Process, Proceeding 5th IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises', IEEE Computer Society Press, 220-225.

NESEIBEH B, KRAMER J and FINKELSTEIN A. (1994). A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification, IEEE Transactions on Software Engineering 20(10), 760 – 773

OBERNDORF P. A., BROWNSWORD L and MORRIS E. (1997b). Workshop on COTS-Based Systems, Software Engineering Institute, Carnegie Mellon University, Special Report CMU/SEI-97-SR-019, November 1997b.

PAULK M.C., CURTIS B., CHRISSIS M. B. and WEBER C.V. (1993). Capability Maturity Model for Software, Version 1.1, CMU/SEI-93-TR-24, Software Engineering Institute, Carnegie-Mellon University.

PLIHON V., RALYTE' J., BENJAMIN A., MAIDEN N.A.M., SUTCLIFFE A. G., DUBOIS E. and HEYMANS P. (1998). A Reuse-Oriented Approach for the Construction of Scenario Based Methods, submitted to ICSSP98 Conference,

RANDELL B, RINGLAND G and WULF B. (1994). Software 2000: A View of The Future, ICL and the Commision of the European Communities, April 1994

SOMMERVILLE I, RODDEN T, SAWYER P, BENTLEY R and TWIDALE M. (1993). Integrating Ethnography into Requirements Engineering Process, Prceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 165-173

SPANOUDAKIS G. and CONSTANATOPOULOS P. (1995). Integrating Specifications: A Similarity Reasoning Approach, Journal of Automated Software Engineering 2(4), 311-342.

STAVRIDOU V. (1997). COTS, Integration and Critical Systems, Proceedings IEE Colloquium on CITS and Safety Critical Systems, IEE Digest 97/013, January 1997

SUTCLIFFE A. (1997). A Technique Combination Approach to Requirements Engineering, Proceedings 3rd IEEE Symposium on Requirements Engineering, IEEE Computer Society Press.

YU E S K. (1993). Modelling Organisations for Information Systems Requirements Engineering, Proceedings of IEEE Symposium on Requirements Engineering, IEEE Computer Society Press, 34 -41.

DAVIES M.A (1999). Making Requirements Management Work for You, Omni-Vista Inc.

CARNEY D. (1999). Requirements and COTS-Based Systems: A Thorny Question Indeed, The COTS Spot, SEIinteractive, Vol.2, Issue 2, June 1999

SWANSON D. B. and McMANUS G. J. (1997). C++ Component Integration Obstacles, TRW, Inc.

Bibliography

Ncube C and Maiden N A. M. 2000, Selecting the Right COTS Software: Why Requirements are Important, to be published as a chapter in a book entitled: Component-Based Software Engineering: Putting the Pieces Together, an Addison-Wesley Longman Publication, summer 2000

Ncube C & Maiden N.A.M. 2000, COTS Software Selection: The Need to make Trade-offs Between Systems Requirements, Architectures and COTS/Components, to appear in the proceedings of ICSE-2000, Limerick, Ireland

Ben Anchor C & Ncube C. 2000, Engineering the PORE Method for COTS Selection with the MAP Process Meta-Model (to appear in the REFSQ*2000, in conjunction with CAiSE 2000,Stockholm, Sweden

Ncube C and Maiden N.A.M. (1999). Guiding Parallel Requirements Acquisition and COTS Software Selection, Proceedings of 4th International Symposium on Requirements Engineering, Limerick, Ireland, June 1999

Ncube C and Maiden N.A.M. (1999). PORE: Procurement Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm, Proceedings of the 2nd International Workshop on Component-Based Software Engineering (held in conjunction with ICSE'99), Los Angels, USA, May 1999.

Maiden N. A .M, James L and Ncube C (1999). Evaluating Large COTS Software Packages: Why Requirements and Use Cases are Important, Proceedings of the 1st International Workshop on Ensuring Successful COTS Development (held in conjunction with ICSE'99), Los Angels, USA, May 1999.

Maiden N.A.M. and Ncube C. (1998). Acquiring Requirements for Commercial Off-The-Shelf Package Selection, *IEEE Software*, 15(2), 46-56.

Ncube C and Maiden N.A.M. (1998). Why Model Software Products: Why, Guiding the Component-Based Information Systems Engineering Process, Of Course!, Proceeding of the CBISE'98 of the CaiSE'98, Pisa, Italy, June 1998.

Maiden N.A.M., Ncube C. and Moore A. (1997). Acquiring Requirements for Commercial Off-The-Shelf Package Selection: Some Lessons Learned, *Communications of the ACM*, 40(12), 21-25.

Ncube C. and Maiden N.A.M. (1997). Procuring Software Systems: Current Problems and Solutions. Presented at 3rd International Workshop on Requirements Engineering, Barcelona, Spain, June 97. CAiSE proceedings.

Glossary

NOTE: Words in *italics* are also defined in the glossary.

Term	Definition
ACRE	**AC**quisition of **Re**quirements Framework that provides methods for acquiring requirements from *stakeholders* and assists requirements engineers to choose methods for *requirements acquisition*.
AHP	**A**nalytical **H**ierarchy **P**rocess is decision-making technique that is based on decomposing a *multi-criteria decision* problem into hierarchy and ranks alternatives through pair-wise comparison between the alternatives.
All products eliminated	All the products that were being considered have been found not meet customer requirements and therefore have been *rejected*.
Atomic functional requirements	Individual requirements that have been decomposed to their lowest level.
Bespoke development	System is developed in-house to provide specific functionality or services.
Candidate product	A product that is being considered as a possible solution.
COTS-Based **D**evelopment (CBD)	Is the process of developing systems using COTS software products. *COTS software products* are bought of the shelf and adapted or integrated to provide system functionality.
Component-**B**ased **S**oftware **E**ngineering (CBSE)	Is the process of building applications from discrete, inter-related *software components* in which applications are developed by integrating existing components.
Complex non-functional requirements	Non-atomic *requirements* that have dependency relationships with other requirements. Achieving each requirement may require achieving many other supporting requirements.
Compliance checking	Determining that a mapping relationship exists between a product feature and a functional requirement.
Compliance mapping	Is a mapping between a problem (i.e. customer requirement) and a potential solution to that problem (i.e. product feature).
Compliance model	Models mapping between one or more customer requirements and one or more product features. It models the relationship between the *product model* and the *requirement model.*
Concurrent requirements acquisition and product selection	*Requirements acquisition* enables product selection and *product selection* informs requirements acquisition therefore resulting in concurrently acquiring requirements and selecting products.
Context-driven process	The next step to be performed in the process is driven by the context of the product model
Contract production process	The process of negotiating a contract with the supplier of the product based on the customer requirements.
Commercial-Off-The-Shelf (COTS) software product	Software products sold, leased or licensed to the general public from a commercial entity in the business of making a profit from the product; with multiple identical copies available to different organisations; where integrators use the product without modification of its internals; and the commercial entity provides product support and evolution
Customer	The person/s paying for the development, and owner of the delivered system.
Customer requirements	Are the needs of the person paying for the system
Degree of compliance	A score showing compliance mapping between requirement and product features. The degree of compliance helps to prioritise the requirements during product selection
Desirable	A requirement or product feature that is not critical to the success of the system.
Discriminating requirements	Those requirements that help to discriminate between candidate products. Help discriminate one product from another in terms of features provided.
Enterprise **R**esource **P**lanning (ERP)	Very large enterprise-wide off-the-shelf integrated packaged application software solutions that require high levels of organisational changes in their implantation.

Essential goals	The goals that each COTS product selection process must achieve in order for it to be a success.
Essential requirements	Those requirements that the system can not do without. There are the core functionality of the system and they are not met, the system would not fulfil the needs of the stakeholders. Essential requirements are mandatory and have a very high priority.
Expert evaluation	Experts in requirements engineering and product selection determine the usefulness and effectiveness of the advice and guidance provided by the proposed method
First-pass requirements	The initial requirements identified by the requirements engineering team and are used to determine the scope of the candidate products to be identified in the market. First-pass requirements are usually very high level and are the core of the user's needs.
Fit criteria	Objective measure for defining the meaning of a requirement, and eventually testing whether a given solution satisfies the original requirement. It is an unambiguous test of whether a solution meets the requirement. An objective measure that will enable testing to determine if the goal has been met by the product.
Fixed-point scenario	Is the idea that the at some point in time all the stakeholders know everything what they want and agree with everyone else and everything will remain the same for the entire duration of the system, e.g. producing a requirements specification and freezing it before starting building the system.
Functional requirement.	An action that the product or system must be able to take, something that the system must do. Functional requirements are the fundamental subject matter of the system.
Generic process model	Is the processes that are often undertaken although it's not necessary to perform all the processes during product procurement
Global constraints	Are the constraints that apply to the system as a whole. For example, the customer for the system is a global constraint, as is the Purpose of the System
Hands-on-evaluation	Evaluation in more detail of shortlisted alternative products in the second stage of the evaluation process during vendor demonstrations using test cases for individual requirements
Inference engine	Is the most important component of the of the process advisor prototype tool. The inference engine represents the controller and is where reasoning about the problem occurs and where decisions are made about what happens next and uses several rulesets to guide the process logic reasoning
Insufficient requirements	The currently identified requirements are not sufficient enough to enable effective product evaluation or discriminating between candidate products.
Iterative process	Iteratively rejects non-compliant products at the same time acquiring detailed customer requirements, therefore resulting in the number of acquired increasing and the number of candidate product considered systematically decreasing
MCDA	Multi-Criteria Decision Aid
Multi-layered process guidance	Provides three levels of process guidance at any point in the process by guiding the team to achieve process goals and which technique to use to solve the current situation.
Non-functional requirements	Are the behavioral properties that the specified functions must have, such as performance, usability
Optional	Not critical, can do without it
Packaged-based development	Building systems by integrating different software packages
Paper evaluation	Evaluation and selection or rejection of products using information provided by suppliers. The information provided by vendors is collected and compared against customer requirements. The most critical high level requirements are used for product-requirements compliance checking.
Paradigm shift	Fundamental changing the way things are done
PORE	Procurement Oriented Requirements Engineering method for guiding

	requirements engineering team in acquiring requirements for selecting off-the-shelf software products.
PORE method box	A component of PORE that recommends techniques, methods and tools to be used during product selection. The method box determine the relevant technique that will be used to solve process situation
Process advisor	The process advisor iteratively provides the team with guidance and intelligently tailor the advice based on the information provided by the *process engine*
Process chunk	The way in which PORE delivers advice and guidance to the requirements engineering team
Process guidance	Advice prescribed by the method to the requirements engineering about what to do next, based on the current state of the compliance model
Process logic engine algorithm	Links together all the components of the process advisor prototype tool. The process engine is responsible for detecting current *situations* and then recommends process advice to the requirements engineering team.
Process triplet	The process advice and guidance as given to the requirements engineering team providing the process goal to be achieved, techniques to use and the situation that provides context.
Procurement	Purchasing the system from another organisation
Product	The system that we are attempting to deliver. This could be a piece of software, the installation of a package, a set of procedures, a piece of hardware, a piece of machinery, a new organisation, or almost anything.
Package acceptance process	Checks the delivered package or system developed from the packages against customer requirements
Product evaluation	The process of determining whether the product has the capabilities needed to solve the problems
Product feature	A capabilities that the product possesses
Product model	A model showing the properties of a software product and its abstract meta concept and their meta relationships.
Requirement	A measurable statement of intent about something that the system must do, or a property that the product must have, or a constraint on the system.
Requirement model	A model showing the attributes of a requirement and its abstract meta relationships
Requirements acquisition	Is the first phase of requirements engineering process, which mines requirements out of documents or elicits requirements from a face-to-face communication between requirements engineer and *stakeholders*.
Requirement dependencies	Other requirements that use the same information, or have a change effect
Requirements Engineering	Is the process of embedding systems within their environment rather than on the prescription of the system's functionality or structure. It is the process of establishing the services the system should provide and the constraints under which it must operate.
Scenario	Is one specific ordering of events, the ordering of which is dependent on the start – and end –events for each action.
Scenario model	Is a model that depicts the sequence of tasks that are performed by a stakeholder. The model describes a sequence of actions and events for a specific case of some generic task which the system is intended to accomplish.
Scenario walkthrough	Is a systematic way of traversing a scenario model and identifying relevant impacts
Simultaneous trade-off	Continuous evaluating the advantages and disadvantages between stakeholder requirements, software products and the system infrastructure
Situation	Is the current state of the compliance model inferred from the properties of the product model and the requirement model.
Situation guidance	The current situation determines what the requirements engineer should do next in the process given as a process triplet
Situation model	Is dynamically constructed model that determines the next process goals and techniques t apply based on the current situation inferred from the compliance model

Situation rules	Rules that infer the current state of the product model and requirement model to determine compliance relationships between *product features* and *requirements*
Software component	Is a unit of software that can be independently deployed and composed without modification according to a composition standard defined by a *software component framework.*
Software component framework	Defines a standard for component composition and interaction and how other entities interact with the component as specified by the component's defined interfaces.
Stakeholder	A person who has an interest in the successful development of the system or other people or organizations who are affected by the system or whose input is needed in order to build the system.
Storyboard	Are pictorial representation of possible ways the stakeholders would use the system. There are extensions of the text-based scenario models and they are used in conjunction with the scenarios to capture what happens and those that are involved in a pictorial way.
Supplier demonstration	Is a detailed product evaluation in which suppliers of shortlisted products are invited to the customer site to conduct a controlled demonstration of their product to the evaluation team.
Template	Systematic way of presenting process high level process guidance and what techniques to use at each stage.
Typical problems	The problems that were experienced by a particular organisation during the process of procuring their software systems
User trial evaluation	A final preferred product is installed at the customer site so that it can be further evaluated in a more representative business environment.