Architecture Assessment of Information-System Families

a practical perspective

Architecture Assessment of Information-System Families

a practical perspective

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. M. Rem, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
vrijdag 1 juni 2001 om 16.00 uur

door

Thomas J. Dolan

geboren te Athlone (Ierland)

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.C. Wortmann

en

prof.dr.Dipl.-Ing. D.K Hammer

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Dolan, Thomas J.

Architecture Assessment of Information-System Families: a practical perspective

by Thomas J Dolan – Eindhoven:

Technische Universiteit Eindhoven, 2001.

Proefschrift

ISBN 90-386-0943-4

NUGI 684

Keywords: Architecture / Information Systems / Architecture Assessment / Product Family / SAAM / Interoperability / Extensibility

Printed by Eindhoven University Press

Cover Illustration: M.C.Escher's "Waterfall" © 2001 Cordon Art B.V. - Baarn - Holland. All rights reserved.

To my parents –
for enabling others to tread the paths denied to themselves.

Preface

He had been eight years upon a project for extracting sunbeams out of cucumbers, which were to be put into phials hermetically sealed, and let out to warm the air in raw inclement summers.

Gulliver's Travels; Jonathan Swift 1667-1745

Despite strong signs to the contrary during the last 6+ years, the old saying is true " all good things (and Ph.D. studies) come to an end ". This research began in November of '94 under the general theme of improving product quality for software-system families, and has since then undergone some refinements, finally focusing on architecture assessment of information system families. Many parties have been involved in the intervening period and I would like to take this opportunity to acknowledge their contributions.

Firstly, I would like to thank Prof. Martin Newby, and the Frits Philips Quality Institute (FPQI), for initiating the research project, and providing the University funding. Unfortunately I lost Martin's academic input since he moved to City University London in 1995; FPQI, however, continued to provide financial input throughout. Prior to Martin's departure, Prof. Hans Wortmann had become, and remained, the main academic sponsor behind the research.

Hans has been of great support to me both during and before this research project. His international co-operation with Jim Browne enabled me to come to Eindhoven to work in 1993, and later to join the I&T group here and start my doctoral research. Hans also supported (sometimes against his own counsel) my attempts to combine working in industry and academia; and was instrumental in enabling this combination and making it bearable. I'd like to thank Hans for his openness in allowing me to orient the research towards my own interests, for his constant positive attitude, and his insightful attention to both research and non-research issues.

The other members of my research committee also deserve thanks. Foremost is Dieter Hammer, who joined the core research team in the middle of the project. Dieter provided encouragement and significant contributions in focusing the research and sharpening my understanding of architecture. Rob Kusters and Aarnout Brombacher receive thanks for their "fresh" input to improve the quality of the later drafts of this manuscript. I'd like to thank Rick Kazman (whose work inspired this research), Jan Bosch, and Wil van der Aalst for having kindly agreed to participate in the reading commission and the graduation ceremony.

My industrial advisors from Philips are thanked for their ongoing support. Peter van den Hamer for his advice, unique reading style, and his improvement-oriented critique. Ruud

Weterings deserves special mention. Ruud has been involved in shaping this research effort from the start and his mark is to be found deep and often in this thesis. His inspiration, humour, and wisdom were needed throughout; they were freely provided, and are appreciated.

To those people in Philips and Baan (now Invensys) who provided opportunity and effort for the case studies used to validate the research, I'd like to express my thanks. Gerardo Daalderop, Stewart Higgins (who introduced me, and many others, to use-cases) and Phil van Liere at Medical Systems were both accommodating and positive in their contributions. At Baan, Peter van Dam was instrumental in implementing the case, supported by Paul Eringfeld and the other assessment participants. Hans Moonen and Martijn Dullaart (TUE) also provided support. I am particularly grateful to Phil and Peter, architects with capital "A".

To my university colleagues at I&T, both old and new, I'd like to say thanks for making my time there enjoyable. Special mention goes to the "old-school": Pierre Breuls, Freek Erens, Monique Jansen, Theo-Jan Renkema, Roel van de Berg, Marcel 't Hart, and Rob van Stekelenborg. I'd also like to thank Karin, Ada, and especially Ineke, for coffee-talk and helping me navigate the stormy seas of professors' agendas. To Naud Vermeulen, and Hermann Hegge who always expressed an interest in my research, and who never expected to hear this – "it's finished". Teade Punter, Bas Vermeer, Rini van Solingen, Remco Helms, Sjaak Bouwman, and Frank Berkers are thanked, just for being there and MP3 files.

I'd like to thank all those in Ireland, Netherlands and elsewhere who helped make life away from research (it did exist) enjoyable and for pretending to understand why I was (still) in college: you know who you are. From this group, thanks to Pat Hoban for proof-reading.

My Dutch family (Vera, Guus and Sjoerd Jan) welcomed me in, and contributed greatly to my general well-being throughout my time in the Netherlands. They deserve thanks not only for this, but also for their concerted effort in the final stages of compiling this thesis especially in formatting, printing and generally sacrificing their weekends.

The biggest debt of gratitude belongs to Anneleen, who has been behind, before and beside me throughout the last eight years. She has provided immeasurable emotional and tangible support in preparing this thesis and it would not have happened without her. I hope that I can repay to her all the lost weekends and holidays of the past years. It's now time for me to stop devoting all my time to software families, and start devoting it to real families!

Tom Dolan
Eindhoven, April 8, 2001

Now my ladder's gone
I must lie down where all the ladders start
In the foul rag-and-bone shop of the heart.
The Circus Animals' Desertion; W.B. Yeats; 1865-1939

Summary

Information processing and the software systems supporting it – information systems – are playing a more prominent role in modern society as information in digital form is increasingly being utilised and exchanged between corporations and individuals.

The growing market opportunities for information systems coupled with the ongoing skills shortage across the entire software-development industry has led to an increasing interest in a product-family approach to information-system development. This approach seeks to apply an industrial and market perspective to information-system development and life-cycle management. From this perspective an information system (or a related set of such systems) is produced efficiently and sold to a market such that the multiple individual customers comprising the market can customise the system to their own situation. Efficiency is achieved through *reuse* based on commonality in the application and technology domains, and customisation is supported through the provision of *flexibility* in the products and services associated with the family. The family approach to product development has proven successful in other industries e.g. the automobile, and contrasts with the project-based, single-customer approach associated with the majority of conventional information-system development. Balancing the forces of flexibility (accommodating change) and reuse (controlling costs) in the dynamic market and technology environments associated with information-system families is a non-trivial challenge.

The work described in this thesis supports the information-system family producer in managing the flexibility required of the family, via leveraging an essential family asset – the architecture. An outline method (FAAM – Family Architecture Assessment Method) is provided enabling the family development team to elicit, analyse, and uncover architectural support for a selection of system qualities associated with important flexibility requirements. The method facilitates ongoing family architecture improvement by enabling a structured dialogue between stakeholders and architect in relation to the selected system qualities, from an early phase in the family life cycle.

The method is based on the pioneering work of the SAAM method [Kazman, *et al*., 1996] in software architecture analysis. SAAM uses stakeholder input to derive requirements and scenarios to illustrate system qualities, in indicating how well an architecture addresses a set of system qualities. SAAM has been adopted by the Software Engineering Institute at Carnegie-Mellon University, and is undergoing active development there and elsewhere in the architecture research community. It is a solid basis on which to found ongoing architecture assessment improvements. FAAM is such an improvement; and it extends SAAM in several important aspects:

- SAAM reporting is characterised by one-off assessment events, with external architecture experts driving the process of assessment, and creating the required artefacts. FAAM is

based on incremental, repeated assessments as part of an ongoing quality improvement strategy and is driven by the family development team itself; so as to embed the practice in the organisation.

- SAAM is a generic method and has paid most attention to describing *what* activities and artefacts must be provided. In order to propagate the *repeatable* and expert-independent practice of architecture assessment to the wider development community, attention must be paid to guiding method users in *how* the activities and artefacts are provided. This is particularly important for flexibility requirements, as there is no broad experience base from which the stakeholders can draw. Providing practical guidelines and techniques in this area is a valuable contribution of FAAM towards increasing the adoption of architecture assessment in industry.
- SAAM has not been extensively applied to the domain of information-system families; most reported experience is on single-system architectures, and addresses the roles of operational stakeholders from the user or development communities. SAAM has also focused on a subset of the important system qualities *viz*. security, adaptability, portability. FAAM contributes to extending architecture assessment to families by actively involving stakeholders concerned with the strategic aspects of the family (e.g. business manager, development manager) and concentrating on the important family flexibilities of interoperability and extensibility. These qualities are examined from a family perspective and practical techniques are provided to generate, specify and analyse requirements in these fields.

The FAAM method has been validated in two industrial case studies, which have been reported herein. These serve not only to test the theory of the method but also provide concrete illustration of how the method is implemented.

Empirical evidence from the case studies supports the relevance and effectiveness of incremental, self-assessment of architecture as a means to improve architectural quality. The stakeholders expected follow-up on proposed changes and requested that repeat assessments would be held for other issues. Architects also supported the method techniques for their ability to uncover both concrete stakeholder requirements and architecture rationale for the, sometimes vague, system qualities of interoperability and extensibility. They reiterated the opinion that architecture assessment provides most long-term value when repeated across projects and families and incorporated into the organisation's overall development process.

FAAM is a first step towards providing practical techniques to support development teams in applying architecture assessment in specific domains (e.g. information-system families), for specific system qualities (e.g. interoperability, extensibility). It is postulated that the approach and experiences reported in this thesis provide a base on which to extend the practical application of architecture assessment to other domains and system qualities.

Contents

Chapter 1
Introduction

1.1 Context

In recent decades there has been an increasing dependence on the generation and exploitation of information (as distinct from material goods) as a means of creating, supporting, and enhancing both existing, and new, economic and social services. Terms such as "Information Technology", "Knowledge Workers", "Information Systems", "Information Economy" have been around for a while, but their increasing adoption in the *lingua franca* of the "common" consumer signifies the spread of information-related services from their origin in government- and financial institutions to the mass-market. Examples such as centralised voice-mail, loyalty cards for customised retail marketing, and cashless payment, attest to the powerful role information processing plays in many aspects of modern life.

As a consequence of the developments mentioned above, the automated[1] information systems which underlie the information services are becoming:

- more *numerous*;
- more *critical* to both producers and consumers;
- more *interdependent* as information is increasingly being exchanged between different parties.

These facts are driving the trend to manage the development and implementation of information systems as an industrial activity so as to realise the efficiencies and quality typical of other well-established industrial products, e.g. cars.

The research reported herein is concerned with the development of information systems in an industrial context. Industrial context in this research means that:

- one organisation (the producer) makes information systems which are sold to multiple separate organisations (the customers);
- the information systems are developed as a "product family" – where market and technical similarities between the systems are leveraged to achieve economies of effort.

In particular, the quality of an important development artefact of the information-system family is examined: the architecture (the set of constituent components, their properties, and interrelationships). The focus will be on how the development team in the producing

[1] Information system is a general term and of course not all aspects of information processing should be automated. However, this work reserves the term "information system" as referring to those parts of the system where software automation is used as the prime means of information processing.

organisation can assess the quality of family architecture. The assessments will concentrate on how well the architecture supports a selection of those general system properties (also called the "ilities") e.g. modifiability, extensibility, which are so important for achieving product-family economies. The remainder of this thesis is devoted to the analysis, design, and testing of a method to perform such assessments on information-system family architectures from the context of an incremental approach to family development.

Firstly, this chapter provides an overview of important issues in the research problem domain, i.e. family based information-system development, and provides initial insight into the terminology introduced above and used throughout this work. Hereafter, Chapter 2 presents the problem statement to be addressed, with the accompanying research design and resulting thesis structure. A rigorous treatment of terminology and underlying concepts is provided in Chapters 3 and 4; in particular these chapters will provide supporting arguments from literature for many of the observations made in this introduction. Chapters 5, 6, and 7 address the design, implementation and evaluation, respectively, of the proposed assessment method.

1.2 Purpose and structure

The purpose of this introduction is to present the reader with an overview of information systems, indicating why a product-family approach to development is appropriate. Hereafter the role of architecture and its importance in information system families is highlighted, and the assessment approach to architecture quality pursued in this research presented. This chapter closes with a summary.

1.3 Information systems as product families

1.3.1 Information systems

In this work "information systems" will refer exclusively to those *software* systems supporting information processing. Information systems are therefore a special case of software system and are defined as (see [Verrijn-Stuart, 1989]) "*Computer-based systems intended to provide recording and supporting services for organisational operation and management*". Information systems are typically associated with data-intensive, operational or administrative processes such as: salary administration, logistics management, and with organisations such as: insurance companies, banks, and government institutions. As mentioned previously, however, the scope of information processing is broadening to diverse areas of the economy, especially manufacturing and services, where their increasing integration is reflected in the term: Enterprise Information Systems (EIS).

The following characteristics of information systems are important for the research domain:

- The information-system customer is typically a professional business owner who depends on the information system to support his[2] primary-process. The information system is thus business critical.
- The producer-customer relationship is (therefore) typically long-term and business-to-business. Significant customer investments are made, for example in user-training, configuring, but also in databases, business processes and information models that cannot be changed easily. Consequently the customer is very influential, and both the producer-customer relationship and the information system itself are long lived.
- The information system supports primary processes for the customer. The customer-specific nature of these processes means that customisation and configuration is necessary both initially, at installation, and throughout the system's life cycle as the business processes and the role of the information system therein, evolve (e.g. re-configuring, extending, porting). Reuse of these initial investments, and the facility to evolve them is important.
- The information system is typically a high-level, domain-specific application built on common "off the shelf" information technology (IT) tools such as: databases, operating systems, network software, and hardware. Typically, some maintenance and extension is done independently of the producer to leverage the experience base of the (reused) common tools.
- The business processes supported by an information system are typically part of a larger value chain involving other customer processes, and even processes involving third parties. Interoperability with other information systems is inherent in the application domain and provides the opportunity for the customer to reuse investments (in information-, and deployment assets) already made in the wider application domain.

These characteristics indicate that information-system customers demand flexible systems, and are concerned with the long-term evolution of the system to support their changing business. They also expect initial investments in modelling, training, and infrastructure to be utilised across system evolutions. Producers, on the other hand, must provide the flexibility to address individual customer requirements, and to do so in a way that addresses multiple customers (a market) in order to grow market share by leveraging domain similarity. Addressing the wishes of a market for customised solutions means that development efficiencies are necessary to control costs. This efficiency is most obviously realised through incorporating the domain similarity into development, and reusing the common aspects across all customers (thereby sharing the investment) in the market.

The fact that information-system producers and customers are concerned with flexibility and reuse over a significant time period (years), makes a product-family approach to development an attractive business model. But what is meant by "a product-family approach"?

[2] The "he" form of personal pronoun is used throughout this thesis for convenience only.

1.3.2 Product families

A product family is defined by Meyer and Lopez, [1995] as *"a set of products that share a common core technology and address a related set of market applications"*. They add that the commonality of technologies and markets leads to efficiencies and effectiveness in manufacturing, distribution, and service, where the firm tailors each general capability to the needs of specific products or niches. Chapter 3 contains more details on product families.

This approach of leveraging commonality while providing variety (to address individual customer wishes) has proven successful in certain sections of industry, the most notable example being the automobile (see [Womack, *et al.*, 1991]). Other common examples of families are mobile telephones, personal computers and televisions – where choice is offered in various user or regional features, yet significant parts of the products (especially the technical foundation) are similar, and reused in all instances.

It is useful at this stage to also discuss what is **not** a product family, as many of the issues discussed above (dealing with varying customisation, enhancement across generations) are common to almost all product development. Rather than present a general framework of what is and is not a product family, some examples will be used to highlight the key issues that distinguish product families from other means of product development.

Some sections of industry are involved in what has been termed one-of-a-kind production [Wortmann, *et al.*, 1997] – where producers design and develop products individually (e.g. shipbuilding, creating marketing campaigns, building corporate headquarters). Such products are termed *single products* and defined [Erens, 1996] as "A *product that has hardly any pre-defined relationship with other products*". The dominant focus is on satisfying the individual requirements of a single customer, without paying attention to reuse of any existing assets (except personnel's skill and experience); this is not product-family development. A less extreme illustration is the finance sector, where a single organisation (e.g. an IS department) may have developed and maintained various financial packages independently as single products. These packages may have overlapping functionality and shared users, yet different interface-paradigms, technical platforms and release cycles. These packages are related, but this relationship is not leveraged in development; again these are not members of a product family.

The key characteristics of a product family (absent from the previous examples) are:

- the co-ordination of the various individual products, especially as regards content and release-timing (i.e. individual product planning accounts for other products in the family);
- the leveraging of commonality so that shared features are developed once and reused across the family.

The family concept of providing *flexibility* to the customer while minimising the associated costs (through repeated *reuse* of common assets) for the producer is appealing to the

information-system industry. However, as indicated in the financial-system example previously, the current information-systems domain is characterised by a single-product approach to system development. Bringing effective product-family practice to information-system development provides many research and practical challenges.

1.4 Architecture and its assessment

The previous discussion on families referred to the importance of the "technical foundation" underlying the family and supporting the business needs for flexibility and reuse. A key element in the foundation of any (software) system is the architecture. The architecture is the manner in which system components are organised and integrated [Zwegers, 1998], and just as in its most familiar incarnation – buildings – architecture determines the development steps, the functions, the structure, and the possibilities of the system. Recalling the previous discussions on information systems and families, it is the latter aspect of architecture – possibilities – that is of most interest in this research work. Both information-system customers and developers are concerned with how to evolve and change the system so that it can e.g.:

- accommodate more users, data, or resources (scalability);
- provide user-specific features and settings (customisability);
- accommodate new features and functions (extensibility);
- co-operate with other information systems in the domain (interoperability);
- be modified to change its behaviour (adaptability);
- run on other computing infrastructures (portability);
- avail of existing own or third party components (reusability).

The general system-wide properties illustrated above are termed "system qualities" [Clements, *et al.*, 1995], and are distinguished from the user-oriented operational functions which also must be provided by systems. Many of the issues addressed as system qualities above were traditionally termed "non-functional requirements"; this misnomer, however, cultivated the impression that e.g. performance was "unrelated" to the function, and also that techniques for functional analysis did not readily apply to these other system properties. In this thesis the term "non-functional requirements" is not used further. The system qualities are more emphasised in product families than single products, because the wider scope of the former in terms of customers, products, and time means that the likelihood, and degree, of changes to be accommodated is greater.

Architecture, because of its determining influence on the qualities of the system, is a key asset of the information-system family (see [Bass, *et al*. 1998]). Architectures, however, are abstractions (the architectural drawing is not the building) and together with the system qualities discussed above, have not been well-addressed in conventional software development methods [Hammer, 1996], [Bennett, 1997]. The quality of the architecture is important, largely ignored during development, and most obvious after systems have been fielded – when it is too late. The scale effects of architecture failures are magnified in a

product-family development environment, making the assurance of architecture quality especially important here. A fuller treatment of software architecture is provided in Chapter 3.

Assuring architectural quality for information-system families is the topic addressed in this research. The proven, and well-established, assessment approach towards instilling quality [Fagan, 1986] is chosen. It is advocated that incremental and iterative assessment of the family architecture with respect to requirements, driven by the family stakeholders, and applied throughout the family's development and maintenance will enable the effective leveraging of architecture to support the family goals. Attention to stakeholders is central to this research work. A stakeholder[3] is an individual, team or organisation (or classes thereof) with interests in or concerns relative to, a system [IEEE 1471, 2000]. The research provides a method and practical techniques to support the family stakeholders in assessing the conformance of the architecture to the required system qualities of the family. The method will be operationalised for two important system qualities mentioned previously: interoperability and extensibility.

1.5 Summary

In the information-systems domain the following observations have been made:

- a family approach to information-system development offers possibilities to bring high levels of flexibility and quality while managing costs;
- family development is a complex, requirements-driven, strategic-focused activity which is not yet well established in conventional information-systems development practice;
- a very important artefact in family-based development is the family architecture which embodies the high-level design strategy supporting the family;
- system qualities which govern how the system accommodates both internal and external change over time (e.g. interoperability, portability, extensibility, reusability) are especially important for information-system families;
- treatment of these system qualities in information-system architecture by current software-development concepts and techniques is disproportionate to their importance;
- early assessment of the architecture's conformance to the quality requirements underpinning the strategic direction of the family provides a mechanism to assure that these qualities will be addressed by the development team.

The provision of such an assessment method (operationalised for interoperability and extensibility) will be addressed in the remainder of this work.

[3] A more detailed treatment of stakeholders is presented in Chapter 4.

Chapter 2
Research focus and approach

2.1 Purpose

The purpose of this chapter is to define the scope and content of the research. That part of the problem domain on which the research will focus is defined, the research aims stated and the resulting research questions necessary to address the aims presented. The general approach to, and design of, the research is discussed; hereafter the detailed structure of the thesis is outlined.

2.2 Problem overview and research motivation

In the previous chapter the problem domain of the research (information-system family development) has been introduced and several important aspects of the domain have been discussed. In particular, it has been shown that support is required by practitioners in fully utilising the family architecture in order to maximise the cohesion between the business-driven requirements aspect and the technology-driven design aspect of the family strategy. This cohesion is central in establishing the balance between flexibility and reuse, indicated in Chapter 1 as a central theme in family development.

As is expanded in Chapter 3, architecture is a very important asset in producing information system families. Architectures inherently support and encourage family-based thinking; they are long term investments in solution strategies and organisational structures, and these investments must be recuperated through the repeated reuse of the architecture to derive multiple product lines. Proper product-family development formalises and strengthens this relationship by explicitly co-ordinating the commercial and architectural policies. The architecture is, therefore, an important asset in the family arsenal and must be treated as such (see Figure 2.1).

The goal of the research described here is to leverage the architecture in product-family development through ensuring its conformance to the family requirements. This is done through extending the proven and accepted concept of assessment to address the important, yet largely neglected, areas of system qualities and architecture. It has been indicated in Chapter 1 that the demands for flexibility and reuse in both the application and development domains are of increasing importance in enabling successful information-system development. Provision of a method for the development team to assess architectural conformance to such stakeholder requirements is an immediate and practical contribution to information-system family development.

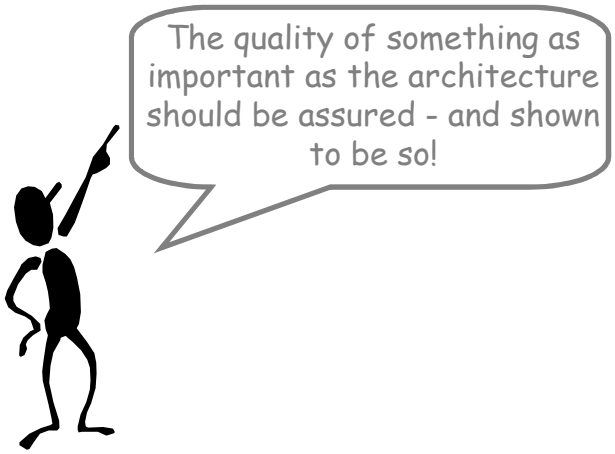


Figure 2.1 : Research Motivation

2.3 Problem statement

Product-family development is a strategically important, communication-intensive activity between multiple disciplines which must consider demands from current, past, and future conditions in defining and developing systems. A design artefact as important as the architecture should support this communication in a way that is both accessible and visible. The major challenges in this area relate to the fact that architecture has tended to be considered as a *black art*, belonging to a small group of individuals in the development organisation. The product-family architecture is:

- the conceptual and technical foundation on which the commercial family is built;
- a representation of high-level design decisions that constrain development;
- the model basis underlying design communication and complexity management throughout the life cycle;
- the basis on which the development organisation is structured.

It is therefore vital that the architecture can be discussed and assessed openly, so as to assure all product-family stakeholders that it meets the requirements placed on it.

The majority of commercial software development methods and tool vendors, however, have concentrated on functional behaviour and have not successfully addressed important issues at the architectural level, particularly those system qualities important for information-system family development. As described in Chapters 3 and 4, the recent past has seen an increasing amount of attention being paid to these issues from the software research community:

- Gilb [1986], for example, has been advocating the cause of system-quality requirements for more than ten years. He advocates the promotion to so-called “attribute specification” to the same importance as functional specification and provides process and document templates to support the approach.

- An assessment approach to architectural analysis (SAAM) has been created by Kazman, *et al.* [1996], and is being extended by researchers at Carnegie Mellon University. This is based on using concrete, stakeholder-generated, usage-oriented illustrations (scenarios) of particular system-quality requirements to uncover architecture support for these requirements via illustrating which architectural components are affected.
- Several research institutes have been considering general product families from a business (see [Womack, *et al*., 1991]) and logistical (see [Hegge 1998]) perspective for some time. These concentrate on how generic products are structured and supported in industrial design and operations processes.
- Both Carnegie Mellon (see [Clements and Northrop, 1999]) and the Fraunhofer institute in Germany (see [DeBaud and Schmid, 1999]) are among those that have recently created initiatives dedicated to software families. These concentrates on providing high-level business process descriptions and techniques, respectively, to support organisations in initiating and supporting product-family based software development.

With all this ongoing attention to various areas of the problem domain, is there anything left to research? The answer is yes. With respect to the central topic of this research – architecture assessment – while the literature on SAAM is comprehensive and the method's success proven, there are some important issues still to be addressed. The SAAM steps are described, but there is insufficient supporting material provided so that the method can be operationalised for use by non-experts; in fact extensive application by non-experts has not been reported, especially as an incremental process within ongoing architecture development. Furthermore, SAAM has not been applied to address those qualities specifically important for system-family development (e.g. composability, interoperability, and extensibility). Extending the method to the system-family environment, where architecture plays such an important role, is a necessary next step.

In general research terms, the above issues lead to the following motivations for further work:

1. Industrial development of information-system families occurs in a context where the various strands of the problem domain mentioned previously (system qualities, architecture assessment, product families) must be integrated. From a research-perspective there is a systems effect – there is more to the whole than the sum of the parts, and integrating the various disparate research directions also warrants dedicated research attention.
2. Apart from the research-integration issue, those actually doing the job of creating, building, and maintaining information-system families also need support in applying the research findings for commercial benefit. Widespread application by the broad practising community is needed to provide the benefits of scientific research to society. Despite the progress made in the various research areas above, their integrated application is non-existent and individual application is small scale and dependent on a few experts or a small number of devotees. It is not sufficient for research to prescribe "what" must be

done and to regard the "how" as simply implementation details. Practitioners are the users of the research community's products, paying attention to users needs is no more (nor no less) than common sense.

The research described herein is concerned with the open research issues indicated above, and concentrates on how to ensure that the information-system family architecture supports some of those system qualities important for the family. The integration of the three separate research domains indicated: system qualities, architecture assessment, (software) product families; is explored with the focus on providing techniques enabling practitioners themselves to incorporate the best of the research efforts in their own environment. This is done based on operationalising and extending SAAM so that non-experts can use it in the context of the ongoing development of an information-system-family architecture. Two system qualities, interoperability and extensibility are used to test the approach. In doing so a significant gap in current quality-assessment research – providing information-system family practitioners with techniques to help themselves in assessing their architectures – is addressed.

2.4 Research aim

The aim of the research work is to develop an outline method (guidelines, metrics, recommendations, and process) for assessing information-system family architectures.

The contribution of the research beyond current architecture assessment methods is in its:
1. explicit attention to, and *active* involvement of, product-family *stakeholders*
2. focus on a selection of family-relevant *system qualities* (interoperability and extensibility)
3. emphasis on practical "*how-to*" mechanisms and *techniques* to enable the development teams within the producing organisations to implement the method.

It is postulated that this self-assessment-based method will facilitate the quicker and deeper adoption of architecture assessment as a means to deliver increased architectural quality. Such assessments provide an early (in the development cycle) opportunity to:
1. address architectural quality (conformance to requirements);
2. provide all stakeholders (including the customer/user) with a shared view of the architecture's progress in development.

Both these benefits help in reducing project or programme risk by:
- providing early (i.e. pre-coding, -testing) feedback on product-family performance;
- building and maintaining stakeholder commitment to the product-family architecture.

It is further claimed that the results of such architecture assessments, and the improved architectural insight gained, will (see also [Abowd, *et al.*, 1997]):
- provide valuable analysis of requirements;
- uncover architectural rationale;
- identify architectural weaknesses (with respect to prioritised family requirements);

- provide a feedback technique to monitor improvements and encourage organisational learning.

These claims are illustrated in the case studies reported in Chapter 6.

The fact that individual system qualities need dedicated treatment is recognised in this research. This is particularly true with respect to developing practical techniques, and two system qualities will be used to provide the necessary focus to the research effort.

All assessment activities are influenced by contextual issues [Punter, 2001], and the outline method and techniques developed here are no exceptions. The expectation is that the overall approach of the method is generally applicable, but the more detailed sub-activities and practical techniques are context sensitive and may require customisation based on organisation cultures, quality/development approaches, and level of family and system-quality maturity. The work reported herein represents initial steps towards generating a design and proving its applicability. Repeated application and extension by practitioners and researchers are needed in order to refine the method to an industrialised process. As indicated above, the method is prototyped by focusing on a specific set of important system qualities, i.e. interoperability and extensibility.

The method is tested in the context of medical, and enterprise information-system family development. It is proposed that the assessment method and the application area can be generalised for use as a practical starting point for other system qualities and application areas, however, the necessary investigation to support this is not addressed in this research.

It is intended that the architectural assessment method will be incorporated into the software product-family development process (perhaps as part of an overall design review prior to detail design and building, and/or in periodic product-wide architecture reviews). This approach can be regarded as an attempt to utilise the notion of the family architecture as a representation of organisational memory, in order to improve organisational performance.

2.5 Research questions

The intention of research questions is to determine which knowledge is necessary to achieve the pre-defined research objective [Verschuren and Doorewaard, 1995]. The following questions must be answered in order to realise the above research aim:

1. **How can the primary stakeholders of information-system families be identified?, and their quality requirements captured?**

In this research a requirements-based approach to quality (quality = conformance to requirements [Crosby, 1979]) is adopted. In order to establish the criteria on which to judge the suitability of the family architecture the important family stakeholders (“customers” of the

assessment method) and their concerns must be determined. The assessment method devotes significant effort to providing mechanisms to help stakeholders in identifying important quality requirements, in particular for the selected qualities of interoperability and extensibility.

Both family stakeholders and architects need means to represent their requirements (the assessment criteria) and designs respectively. Due to the facts that:

- family system-quality requirements are typically more implicit than explicit in current practice;
- development methods are weak in dealing with the system qualities important for product families;

it is a non-trivial issue to determine how the stakeholders can represent and communicate their requirements.

2. **How can the architecture be assessed with respect to the system-quality requirements?**

It is important to provide the assessment participants with a process and supporting techniques so that the assessment itself can be conducted in an optimal manner. The research will provide such a process and accompanying guidelines for structuring the assessment. This enables a common dialogue to occur between participants so that the assessment demands and results can be successfully communicated. Here also the representation issue is important – architecture descriptions are typically based on some variant of the informal "box-and-line" theme, and there is no established practice of presenting them to business-oriented stakeholders. In Chapter 3 the state of the art in architecture practice is reviewed and a technique selected for representation in the method.

3. **How can the results of the assessment be recorded and communicated back to the stakeholders?**

Communicating the results of the assessment back to the stakeholders in an understandable form (i.e. identifying the consequences for their particular concerns), and identifying follow-up actions is a very important part of the assessment. The proposed method will provide advice and concepts to support this feedback activity.

In addition to these method-specific questions, there are also some important underlying issues related to the architecture assessment domain addressed in the research. In particular, contributions are made towards the following aspects, indicated as important from literature or from observations reported in Chapter 1:

- clarifying the plethora of terminology (for examples and discussion see [Bennett, 1997], [Bass, *et al*., 1998]) related to the relatively immature (scientifically) domains of software

product families and software architecture. An agreed terminology is a basic necessity for scientific discourse, and this work offers an extensive overview and synthesis of terminology in Chapters 3 and 4.

- explicitly incorporating non-technical stakeholders intensively in the architecture development process. The position adopted in this research is that the family stakeholders are the "customers" of the family architecture – the architecture is made *solely* to address the business goals, using the technical resources and constraints of the organisation. Chapter 1 has illustrated the paradoxical situation that although information systems and their development are becoming more strategically important for producers and customers, the associated strategic aspects and stakeholders are largely ignored by commercial development methods.
- explicitly incorporating the system-quality requirements as part of the architecture-development cycle. This is related to the previous point; the architecture is largely determinant on the attainment of system qualities (see e.g. [Clements, *et al*., 1995]), yet their explicit treatment is missing from most architecture descriptions [Bennett, 1997].
- encouraging architects to make their work available for assessment by the stakeholders. Software development is as much a social process as a technical process [DeMarco and Lister, 1987]; and any "how-to"-oriented method should address this aspect. Assessment has connotations of judgement and people naturally have reluctance to have their work assessed (see e.g. [Grady and Caswell, 1987] Ch. 7, [Grady, 1992] Ch. 10). The research confronts this issue and provides implementation guidelines, refined through practice, to address it.

2.6 Research approach and method

2.6.1 Research approach

The problem to be addressed in this research is one of **how** to support (in the form of an architecture assessment method) **practising professionals** in **developing** information-system family architectures. The key words from a research methodology viewpoint have been indicated in **boldface** and are discussed below:

How – This research is concerned with providing a process (how-to) to achieve an operational goal – it is therefore best characterised as applied science. It does not *primarily* deal with theory developing "what"[1] questions (see e.g. [Van der Zwaan, 1990] pp 29-53), although theory extension is addressed where necessary. The primary aim is to generate design

[1] This research has a "how" focus which is not *in-vogue* with most mainstream research and its accompanying high regard for "what"-focused questions. While the "what" questions undoubtedly underpin all scientific knowledge, unless the broad research community devote some (more) time to the "how" questions the industrial community will continue to prefix "so" to the "what" research, and the research results will remain largely interesting only within the research community.

knowledge (design models and heuristic statements) [Van Aken, 1994a] that can be used to inform other situations.

Practising – This research is grounded in industrial practice. While product-family development is seen as a promising solution to the problem of meeting the variety and cost demands from the market, there is, however, a distinct lack of a *useful* method which prescribes how a company can successfully apply and sustain product-family oriented development. This research is focused on one key aspect, i.e. architecture assessment, of such a method. The research contribution is developed largely for and from the trenches of industrial practice, where the researcher has adopted the position of *reflective practitioner* [Schön, 1983]. The aim is to provide architecture assessment techniques to those developing families so that they can harness the benefits of this practice independently of scarce, expensive, and external experts, and make it part of *their* normal development process.

Professionals – A dominant theme in this research is that the people actually involved in family development are in the best (only!) position to bring substantial improvement to the overall state of practice. Product families in general and information-system families in particular, are complex entities with high levels of uniqueness between industries and organisations. To aim for a generic assessment method that applies to all eventualities is naïve. A more realistic approach is to provide guidelines and examples of such an outline method in a focused, yet representative, situation and use this as a "prototype pattern" (with all due respects to [Gamma, *et al.,* 1995]) for further extension and customisation by the practising community. This approach is appropriate for the domain of professional practice examined in this research, which values the provision of specific design knowledge [Van Aken, 1994b] that can be reused and adapted according to local circumstances.

Developing – The assessment method proposed herein will operate in the context of an ongoing family-development cycle. Because industrial organisations and individuals are oriented towards delivering products using "standard" development methods, this research will (where appropriate) attempt to leverage those best practices of current software-development methods. This has the advantages of reducing the risk and effort of "re-inventing the wheel" and it also aids industrial acceptance of the proposed method because it is built on familiar "ingredients".

Considering the key research characteristics above – this research is best classified as: ***design-oriented action-research***; in that:

- the research output is an outline method (a set of guiding procedures to be used in the development organisation) and the main research task is the *design* of such a method;
- there is a clear emphasis on *applying* theory to influence *real-life* phenomena and *extending theory* through exposure to *reality*;

- the researcher is a *member* of the organisation that is used to develop and test the ideas, and where one of the case studies is carried out. In this sense the aims of the organisation and the researcher are similar, and both are *jointly responsible* for the research result.

Further, action-research is suited to a socially intensive situation such as information-system development and use [Jönsson, 1991], and has been defined [Argyris, *et al.*, 1985] as "*Action science is an inquiry into how human beings design and implement action in relation to one another. Hence it is a science of practice…a process of critical inquiry…and reflective learning*".

2.6.2 Research method

The case study method of research validation is the most appropriate for this design-oriented research [Yin, 1994]. Generality of research findings is an important aspect when evaluating doctoral research and this research aims to produce generalised knowledge. Due to the perceived weakness of case studies in providing generalised findings, careful attention is given in case-study design (detailed in Chapter 6) towards ensuring the generality of the problem, and the solution within the research domain. There are limitations, however, which must be realised; action-research does not aspire to the levels of law-based generality typical of positivist research (e.g. natural science), and is more suited to the context-sensitive approach of interpretative research (e.g. social science), and to inductive low-level theory generation [Jönsson, 1991]. Validity of research findings is, therefore, strongly based on logical argument and the provision of generalisable design knowledge [Van der Zwaan, 1998] which must be reviewed by experts and interpreted by users to their own situation.

That said, the remainder of this section contains details on the steps to support the scientific value of the case-study results in this research. The general applicability of case study based research[2] is based on *analytic generalisation* rather than *statistical generalisation* [Yin, 1994] and is primarily influenced by the *validity* (internal and external) and the *reliability* (repeatability) of the results. Each of these aspects are dealt with in turn, including an outline of the strategies pursued in this research to address them:

- *Internal validity* – refers to the correctness of the causal relations observed within the individual case study. The strategies employed to ensure no spurious events corrupt conclusions include the use of:
  - multiple units of analysis to provide a broad range of data collection and discussion points to counterbalance any point-disturbances (this is termed a nested case design);
  - well-documented, and clearly operationalised, case procedures – the assessment method design is detailed (see Chapter 5) and is used as the case-study protocol.
- *External validity* – refers to the domain in which the cases' findings are applicable. The research domain has been identified as information-system family development, the following strategies are used to ensure the requisite domain generalisation:

[2] See Yin [1994] Chapter 2 for a thorough discussion on this topic.

  - multiple case studies (medical case and planning case) are used across separate organisations;
  - separate individuals and products are used to provide variety among the cases and so explore a more representative view on the domain.
- *Reliability* – refers to the repeatability of the results, independent of the investigator, in the cases. It is important to realise that this refers to repeating the *exact same case*, with another investigator, not replicating the results with *another case*. The strategy used here is to have a well-documented case procedure; the detail of the method design facilitates this and extensive reporting has been done.

The use of multiple, nested cases and the detailed operationalisation of the case protocol follows recommended practice [Yin, 1994] to support generalisability of case study results. The provision of such *generalisable* knowledge, with well-documented context and procedures serves as a basis for either direct or modified application in related design settings. The method developed in this research provides initial steps for operationalised self-assessment of interoperability and extensibility, and a design framework to extend this to other qualities and domains.

2.7 Research design and thesis structure

The research design is based on the regulative research cycle [Van Strien, 1986], comprising the phases of: problem description, analysis, design, implement change and evaluation, with each phase having a dedicated thesis chapter as indicated in Figure 2.2. This research design is suited to practice-based learning and action research. The regulative cycle is extended with the reflective cycle [Van Aken, 1994a], in order to abstract generic knowledge. The figure also indicates some key features of the research activities associated with each chapter, providing a high-level overview of the thesis structure. The remainder of this section provides some details on the contents of the individual chapters.

Ch. 3 [Problem Domain] – The problem area is described using practical experience, relevant theory and terminology from a survey of appropriate literature in the areas of product families, information systems and architecture. The aim here is to provide sufficient information to appreciate the relevant (for the research) forces in the problem domain and to clearly define the problem to be addressed in its context.

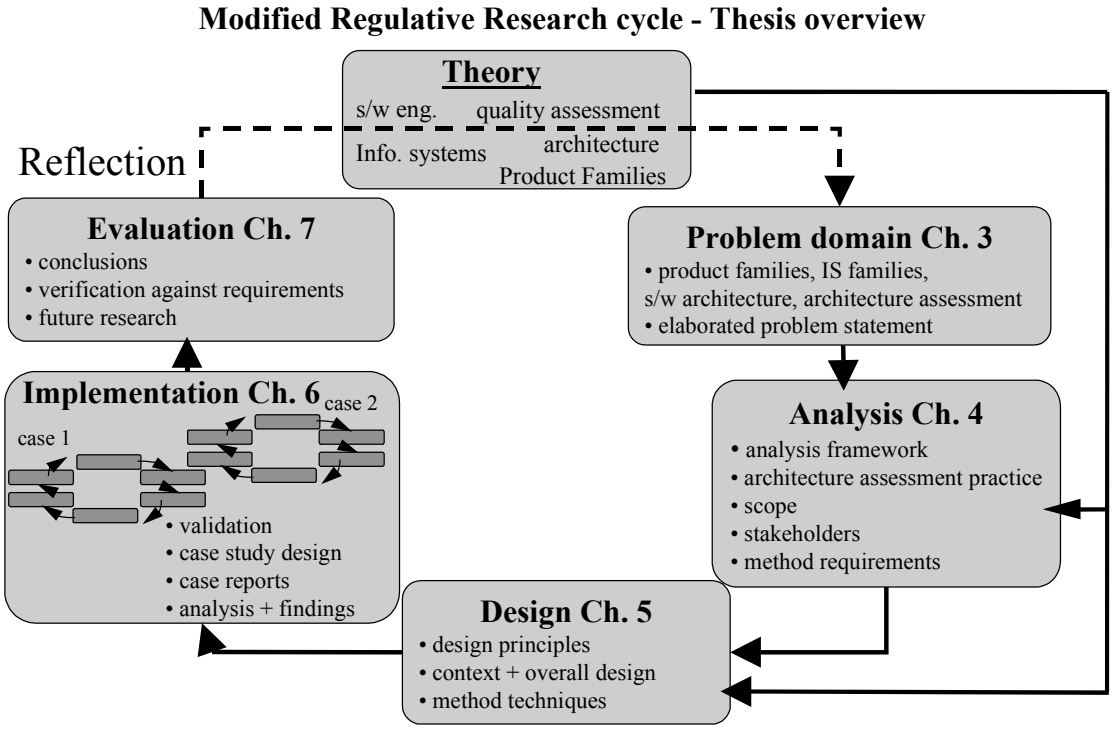


Figure 2.2: Research Design – adapted from regulative cycle [Van Strien, 1986]

Ch. 4 [Analysis] – In this phase the core topic of the research (architecture assessment of family-relevant system qualities) is analysed in more detail. A framework for the analysis is developed, and used to position the focus of the research on strategic stakeholders. The necessary scoping of the research (to concentrate on interoperability and extensibility) and the requirements for the assessment method are also derived. The analysis phase also addresses the first research question (section 2.5): who are information-system family stakeholders, and how do we capture requirements from them? The methods applied here are again literature survey and deduction, normal practices in any problem analysis.

Ch. 5 [Design] – This phase is concerned with the initial overview of the assessment method. The main theoretical and practical components (particularly the use of quantitative and qualitative techniques for requirements elicitation and representation) of the method are explained and the various components synthesised into an initial method definition. In this phase the remaining research questions (assessment execution and reporting) will receive attention. The method is described, concentrating on the people, activities, artefacts and supporting tools involved. This design phase of the study uses literature to provide a base method, which is extended based on the results of the analysis phase and prototyping with users. This incremental approach is typical of industrial design practice and research (see e.g. [Bertrand and Wortmann, 1981]).

Ch. 6 [Implementation] – The implementation phase of the research is concerned with the actual implementation of the method in representative case studies. Two cases, addressing the selected qualities of interoperability and extensibility respectively, from a medium-size

medical information system, and an enterprise-wide business system are reported. The case study method is used to actively develop and generate feedback on the emerging assessment method. The last two research questions (section 2.5) are revisited and refined here in prototyping the proposed method. Case-evaluation with respect to the design requirements is used to refine the method and provide implementation guidelines.

Ch. 7 [Evaluation] – Evaluation of the research is based on reflection (the reflective step) and judgement of the results of the implementation. The evaluation method used in this phase is that of comparing findings against requirements (from quality theory). This is supplemented by expert review and anecdotal "testimony" by involved stakeholders. The general lessons learned are particularly important and guidelines are provided towards repeating and customising the method for other organisations and industries. The thesis is closed with some recommendations for further research in the problem domain.

2.8 Summary

The research problem to be addressed in this research has been summarised and provides context for the research motivation and aim. The resultant research questions necessary to address the aim have been posed, and the action-based nature of the research discussed. Finally, the overall research design and its relation to the research questions and the subsequent thesis structure have been presented.

Chapter 3
Domain – information-system families

3.1 Purpose

The purpose of this chapter is to expand on the description of the problem domain presented in the initial chapters. The focus of the expansion is to:

1. refine the terminology so that a representative and consistent discourse can occur within the thesis (and hopefully without);
2. present a comprehensive review of the main concepts underlying the research, in particular providing rationale for the observations made in the initial chapters.
3. highlight the state of the art in both academia and industry with respect to current practice, accepted challenges, and promising solutions in developing information-system families.

The review will conclude with an elaborated problem statement providing more detail and direction to the initial problem as defined in the previous chapter.

3.2 Structure

The chapter is predominantly based on a review of published literature from academic- and professional-oriented publications addressing the key aspects of the problem domain. The chapter is organised (see Figure 3.1) around the key components of the problem domain as described in Chapter 1. Initially, product families are defined, their role in addressing variety in a cost-effective way described, and their essential characteristics presented. Hereafter, software-system families are introduced; they can be regarded as a special case of product families. The key differences between software-system families and general product families are emphasised. Subsequently information-system families, a subset of software-system families, and their distinguishing characteristics are presented. Throughout, the role of architecture is highlighted to give the reader an insight into the importance of this artefact for all of the families. This leads naturally to the need for a fuller examination of software architecture, outlining the varied treatment it has received in the software research community. In closing the review the current status of the relatively new practice of architecture assessment is presented.

These aspects are presented individually (see heading references in Figure 3.1), with interesting associations between aspects introduced gradually as the chapter progresses.

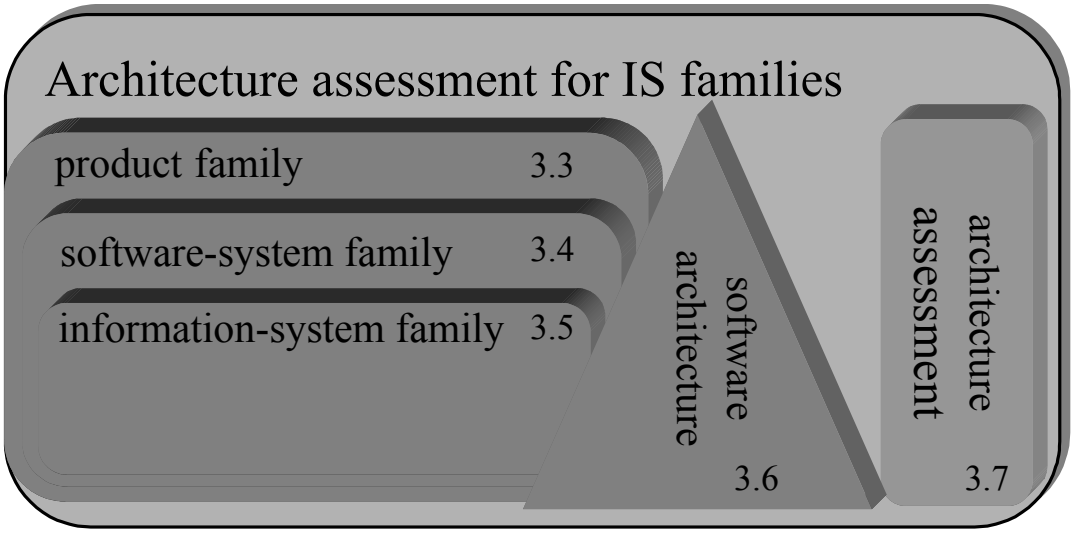


Figure 3.1: Overview of main research themes

3.3 Product families

Product families have been introduced earlier in Chapter 1 of this thesis – the topic is covered in more detail in this section.

3.3.1 What?

The important aspect of product families – accommodating specific needs within the general product framework – is made explicit by Erens [1996], when defining a product family as a "*product concept that is designed for a market but caters for the individual wishes of customers by introducing variety within a defined product architecture…* ".

This emphasises the important fact that product families must be defined based on similar market features *and* a similar technical foundation (i.e. the architecture) supporting those features. The initial presence of such a defined technical foundation generally means that product families require the existence of a relatively mature product-market combination, where all involved parties have a good understanding of the market and technical basis of the product family [Wortmann, *et al.*, 1997].

Information Box 3-A: product families – a common phenomenon

Product families are not rare, in fact most products that we deal with as mass-market consumers are either explicitly produced as families or are candidates for same. The automobile (referenced earlier in this work) is widely used as the textbook example, but there are many others:

- clothing and footwear is produced in many shades and sizes reusing the same design, material-type and production process. Other examples can also be observed from the consumer-goods market;
- more contemporary (if less obvious) examples are the various TV game shows such as "Who Wants to be a Millionaire?", "Big Brother", and "Survivor". Each of these programme formats represents a reusable framework which is sold (or licensed) to different TV networks and can be

customised to their individual language and cultural circumstances, but also retains certain common, core aspects that cannot be changed.

The typical product family is a mass-market product, relatively independent of other families, providing (producer) defined variability, and involving significant production resources. Large markets and large production investments are the main motivators for the explicit management of the flexibility offered and the reuse pursued in the typical product family. The scale consequences of failures are so high that family design and production must be tightly managed. This is contrasted (see section 3.5) with the situation in the professional markets serviced by information-system vendors.

3.3.2 Why?

Families are an approach by producers to meet the increasing pressure from the dominant buyers-market for customised products at mass-market prices [Pine, 1993]. In terms of Porter's [1980] competitive framework; product families, combining customer choice and large development reuse, are used to compete on both the cost and variety axes. This "win-win" situation in the traditional trade-off between variety and cost has made product families an attractive business strategy in contemporary economic conditions.

Another very important aspect of product-family development is time. This has received most emphasis in the so-called "product line" research effort at the Software Engineering Institute (SEI) [SEI product line, web], and the work of Sanchez on "strategic product design". It is concerned with the long-term, strategic issues relating to the leveraging of current product-design investments across other family members (products) and indeed future generations of the family. In particular, product-family thinking seeks to reuse some component[1] designs across multiple products, and across successive generations of products [Sanchez, 1996] to:

- realise the economies of scale necessary to reduce such component manufacturing and development costs;
- increase component reliability through experience.

This recognition that future products will be used to recover initial development investments, means that issues of product scalability and upgradeability must also be addressed by product-family management. The long-term nature of family development has also implications for the variety offered to the market. Customers want products which will provide them with a *future-proof* investment – they want the current product (and its future upgrades) to provide the flexibility to accommodate their own strategic plans. In many cases this means that products must provide standard interfaces to their environment (facilitating integration) and must contain standard technology (thus providing the requisite flexibility to enable

[1] "Component" is a loaded term in the software-engineering literature. In this thesis a "software component" is "*a unit of composition with contractually specified interfaces and explicit context dependencies only…(it) can be deployed independently and is subject to composition by third parties*." [Szyperski, 1998]. All other uses of the term component will refer to its English language meaning: "part of a larger whole" [Oxford English Dictionary].

independent customisation and service). As will be discussed later, this life-cycle management aspect of product families is especially prominent in the case of software.

3.3.3 How?

Several industries have been relatively successful in the application of product-family thinking to their business; the automotive industry is the shining example. Cars are produced in large numbers with lots of choice in end-user features, but with many physical components, and design principles, reused within and across particular models. This production environment in which a large set of products is manufactured in small volumes, but with the scale economies of mass production has been termed *mass customisation* [Womack, *et al*., 1991].

In some economic sectors, however, the majority of family development is *ad hoc*; variants within the (commercial) family are developed as single products and added to the family. This is especially true in the software-systems industry which is less mature, and has less affinity with the manufacturing-oriented benefits and direction of mainstream product-family business. Typically, there is a distinct absence of coherent, proactive family development [Erens, 1996].

One of the key assets in successful product-family competition is the product-family architecture, which enables co-ordination of the overall business strategy (for the family) with the individual product development projects [Erens and Verhulst, 1997] through which the family is realised. In particular the architecture, specifying the components and interfaces at the heart of the family, is responsible for the degree of design modularity in the family and therefore constrains the amount and scope of both reuse and flexibility.

Families are strategically important to the firm because they are a crystallisation of competitive strategy, and a significant commitment of cost and focus (particularly because of their long-term nature). Within the organisation of the family producer itself, there are multiple parties with specific roles, and hence a vested interest, in the family development process; these parties are termed the family stakeholders. The product family is developed and maintained through the *co-ordinated* efforts of the various stakeholders. Because of the far-reaching effects of family decisions on the organisation, and consequently on the stakeholders – it is important that management of the family involves all the affected stakeholders, especially those representing the relatively large customer and user population from the market, and other external stakeholders.

Regarding concrete advice on how families are managed in industry, the complex and confidential nature of this information has meant that such literature is rare; companies are generally left to their own devices. Academia has offered some general advice on the issue, readers are referred to [Erens, 1996], [Erens and Verhulst, 1997], [Bass, *et al*., 1998].

3.3.4 Challenges

The following general characteristics of product-family development can be derived from the previous sections:

1. product-family producers are *market oriented* rather than (single) *customer oriented*;[2]
2. the provision of flexibility (to supply variety) within a reuse (to cover costs) context is the watchword of product-family production;
3. the strategic alignment of commercial and technical product-family definitions is essential for success.

These characteristics contribute to the following challenges for the product-family producer:

- The market orientation of product families means that the organisation is directed towards satisfying multiple customers. Such organisations typically direct their sales, development and support activities by using *internal* representatives of the multiple *external* customers and users. The development activity must support all aspects of the system's life cycle; and therefore must *incorporate* the various stakeholders and their different requirements, and support their communication [Kusters, *et al*., 1999].
- Product families must provide both flexibility and reuse; and represent a balancing-act between a customised product and a mass-produced standard product. The decision on where to be flexible, where to reuse, and how much of each to provide, is captured in the architecture. This is typically complex and demands inclusive analysis and acceptance by all stakeholders.
- The successful alignment of the commercial and technical family definition must be carried out in the context of multiple stakeholders and their extensive communication. The product-*family architecture* is the set of generic technical components and their interrelationships underlying the family. They must support the complex family-development activities of the various stakeholders (after [Clements, *et al.*, 1995]).

Summarising, as with most aspects of management, the success of family management is largely dependent on the people responsible i.e. the family stakeholders. The major challenges in product-family development is in supporting the interdisciplinary discourse between the stakeholders so that they can communicate and co-ordinate effectively. Support is especially needed to facilitate discussion of those strategic-level product aspects particularly relevant for families; e.g. addressing customisation, product longevity, asset reuse across variants and generations.

3.3.5 Potential solutions

Each stakeholder has his own view on the product family. Each view can be seen as a description (model) of the product from a particular perspective, with certain features

[2] This does not mean that family producers ignore customers; on the contrary they are highly customer focused; but families are designed and built for multiple, *a-priori* unknown, customers (a market) in contrast to a single product whose development is driven by an individual customer order. Single products may be derived from a family.

abstracted. These product features are regarded as important by the stakeholder in carrying out his function as part of, and at various points in, the overall development process. Users (representatives), for example, may be most interested in the various functions provided by a product and the relationships between them when specifying requirements. Logistics personnel, on the other hand, are most interested in the physical aspects of the product and how the various parts are assembled together when planning manufacturing. Multiple different views, therefore, are a consequence of the different phases [Van den Hamer and Lepoeter, 1996] and disciplines required to develop a product (family). Exchange of information between the various stakeholders – across the views – is important for successful family development [Danko and Prinz, 1989]; and the architecture, where many of the various stakeholder concerns are integrated, is an artefact to facilitate this exchange.

Architecture has been widely cited as an integral part of successful family development [Erens and Verhulst, 1997], [Clements*, et al.*, 1995] particularly in the early phases of the family life cycle when many of the most important and long-lasting decisions are made. It offers potential to act as an important communication medium for design decisions [Clements and Northrop, 1996] however, work is still needed to ensure that the family stakeholders can provide relevant input and extract relevant output from the architecture. This potential and its realisation is the main purpose of this thesis.

The role of software in providing flexibility in traditionally non-software product families is an increasing trend, and very obvious in that family favourite, the automobile. Options such as ABS braking, route direction services, optimum fuel delivery control, fault diagnosis and event registration are some of the many features providing variety in modern cars which are significantly dependent on software. The increasing prevalence of software, and the emergence of software-system families is discussed in the following section.

3.4 Software-system families

The principles of general product families have been discussed above. This section deals with the family aspects particular to those cases where the product is a software system. Initially software systems will be described, and the fact that software emphasises many conventional family aspects will be presented. Hereafter, certain characteristics of software systems will be discussed and their consequences for software-system families considered.

3.4.1 What?

A system is defined [Rechtin, 1991] as "*a set of different elements so connected or related as to perform a unique function not performable by the elements alone"*. This broad definition allows a product composed of individual elements to be regarded as a special type of system; and such product-related systems can also be regarded as candidates for so-called system families. Furthermore, a *software system* can be regarded as a specialisation of a more general system, where some of the system elements comprise software, and this (software) aspect of

the system is important for the individual dealing with the system. Thus: a software system is *a system containing significant (from the point of view of those involved with the system) software elements*; and where software systems are developed according to the product-family characteristics outlined earlier they are termed *software-system families*. In the remainder of this thesis, any statements made regarding product families will apply to software-system families unless otherwise indicated.

Definition:
The product-line[3] research effort at SEI is concerned with software-system products and has produced the following definition [Clements and Northrop, 1999] of ***software-system families***: *a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or customer; built from a common set of core assets*.

3.4.1.1 Categories of families – software-system tribes

The following categorisation of software-system families [Jacobson, *et al*., 1997] illustrates how families arise in practice:

1. *Suite* of applications which work together to satisfy related business-processes e.g. MS office, banking software;
2. Single (large) applications come in *variants* e.g. Ericsson AXE telephone switching systems have customised features per country and per size (S,M,L);
3. Independent applications built from a *common component* base e.g. MicroSoft Foundation Classes (MSFC).

The first two categories are typical of the general population of commercial software and display the characteristics of market and technical similarity emphasised by conventional family development. Large scale component producers, the third category, are few and will not receive attention in this research.

The second category, *application variants*, is most closely associated with the conventional (non-software) product families discussed earlier in the chapter. Systems in this category are relatively homogeneous and are often monolithic, providing a complete closely related set of functions. The first category, *application suite*, is less associated with conventional product families, and is distinguished from the application variants by its more modular, interconnected nature, and the attention paid to future roadmapping and migration across the family members comprising the suite.

[3] Product lines are products sharing related market features. The real benefits of the similarity is achieved when the products are made using common (development) assets - i.e. made as a product family. The SEI concentrates on product lines made as a product family, so they regard the terms product lines and product families as equivalent.

As discussed later (see Chapter 4), modern information systems increasingly belong to this first category. A trend, reflected in the planning case (see Chapter 6 and Appendix A), is the conversion of many application variant families to the application suite typology. This is motivated by the:

- commercial flexibility provided by the latter as it allows customers to "mix 'n match" between best-of-breed suppliers of partial functionality;
- development flexibility provided through allowing producers to incorporate multi-vendor, third-party components in their families.

The development flexibility in particular, is important in that producers can leverage it to outsource development effort, acquire external expertise, and market their own partial functionality as components in other vendors solutions. More information on these developments can be found in the Commercial Off The Shelf (COTS) (e.g. [Kontio, 1996]) and Component Based Software Engineering (CBSE) (e.g. [Szyperski, 1998]) literature.

The application suite category of software-system family will receive most attention in this thesis.

3.4.2 Why?

The inherent flexibility of software is one of the main reasons for the incorporation of software elements in previously non-software products. Software development is expensive, and the increasing inclusion of software in everyday products (e.g. cars, climate-control systems, phones) has led to a skills shortage in the industry. Companies are trying to recover these costs and increase productivity by adopting the product-family approach to software development.

Furthermore, from the market perspective many business entities (e.g. research hospitals) that had employed their own "in house" software development staff are realising that software development and maintenance are not their core competencies. They see that chasing rapidly evolving technology and fighting for resources in the labour market are burdens they do not want to bear; they prefer to buy their systems rather then make them. Buying a commercial software system from a producer with an established market share also means that they get a "standard" (instead of a one-off) product, and are part of a larger group of buyers with which to share risk and leverage negotiating power with the producer. A family approach to developing software systems makes sense for both the producer and the customer.

3.4.3 How?

General industry-wide application of component reuse to drive the product-family development of software products is not yet established [Szyperski, 1998], [Booch, 1993] despite the pleas from the software-component reuse movement on the benefits of same since the late '60s (see [McIlroy, 1968]). However there are signs that the software industry is more interested in the industrial approach to software typified by families – experiences from the

world of telecommunications [Jacobson, *et al*., 1997], and naval defence [Brownsword and Clements, 1996], [Bass, *et al*., 1998] have been reported. Similarly, conferences and research programmes (e.g. [Clements and Northrop, 1999]) dedicated to software product families are increasing in number and prominence.

Regarding practical advice on meeting the challenges of developing software-system families, the situation is similar to that for general product families as discussed in the previous section. Some promising developments are:

- the previously mentioned product line research initiative of the SEI, [SEI product line, web] which combines research and industrial parties in gathering and dispersing best practice (see e.g. [Bass, *et al.,* 1997, 1999]. One of the most notable contributions is the product line practice framework which summarises the activities needed to implement a software product line development process [Clements and Northrop, 1999].
- The PULSE approach (see e.g. [DeBaud and Schmid, 1999]) from the Fraunhofer institute, provides a method, with tooling, that supports the complete family life cycle. It concentrates, in particular, on the business case underlying the definition of the product family. The method is still largely confined to the research domain, however some commercial exploitation is beginning.
- The FAST (Family-oriented Abstraction, Specification, and Translation) process developed over a decade at Lucent [Weiss & Lai, 1999] provides a comprehensive process for introducing family-based development. It illustrates (among others) that although the concepts underlying families have been around for a while their implementation is not easy, and involves substantial organisational impact.

While the above initiatives are promising, their reported industrial application is still limited (typically 5 – 10 industrial partners are represented at the annual SEI product-line practice workshops). There may be a (very) silent majority adopting these *public* practices, but most likely individual companies have developed home-grown solutions to creating and managing software-system families. Such companies are not, however, sitting on their laurels, and are eager to learn more, based on industrial interest at the various product-family conferences (see e.g. [Van der Linden, 1998, 2000]). The "how?" of software-system families is not a rhetorical question.

3.4.4 Challenges

Perhaps the largest challenge towards implementing product-family practices in software development is historical. The fact that development costs dominate manufacturing costs in software means that management is not amenable towards looking for savings due to similarity in software. Where manufacturing costs and volume dominate (e.g. the car industry), management will seek to reduce these costs through such obvious scale economies as part reuse and standard fittings. Software has never been driven by such obvious manufacturing costs, and cost reduction in a nebulous entity such as design and development is less obvious. The trend is changing, however, and development productivity is under

increasing examination as growing numbers of producers apply industrial approaches to software development (e.g. CMM process measurement, [Humphrey, 1989]).

Management attitude apart, from the standpoint of flexibility and reuse the following challenges are particularly important (but not exclusive) to software-system family development:

- The widely held perception that software is ultra-flexible means that the expectations of users and customers as regards the flexibility, that must be provided by the family, are very high. For example, people generally do not actually move their house when "moving house"; but when computing "foundations" change, software must (because it can) be ported.
- Software systems increasingly find application in environments where interoperability with other software systems is expected. This means that software producers must adopt a *systems-within-systems* view of their product, and must plan for standard interfacing to relevant external systems (this is also termed an *open-systems* approach).
- An increasingly obvious (generic) *customer* needs the facility that he carries out permanent, producer-independent modifications to the software system after initial purchase and installation. This need is not as great in non-software systems, where customisation is essentially the responsibility of the producer, and the customer regards the system as a black box. This fact is illustrated when one considers that customers generally expect non-software systems (e.g. a Deutz-Fahr tractor) to be *replaced*, whereas they expect software systems (e.g. those infamous COBOL-based financial systems) to be *upgraded*, thus preserving their own individual customisations.
- The demand (from both producer and customer) to recoup investments across system upgrades, coupled with the dominance of development over production costs in software, means that reuse *across* generations is more emphasised in software systems. Software reuse and family-oriented development are regarded [Jacobson, *et al.*, 1997] as going hand-in-hand. The challenge of software reuse in software-system family development arises from the immaturity (lack of professional consensus to develop industry-standard components) and flexibility (explosion of customised solutions) of software, which has militated against the off-the-shelf, catalogue-based development typical of non-software-component systems.

Some key aspects of software-system families gathered from the discussion so far are:

- a broader range of flexibilities (also termed system qualities [Clements, *et al.*, 1995], or "emergent properties" [Simon, 1981]) is demanded from software-system families than from single products; by both the customer (e.g. customisability, extensibility, interoperability) and the producer (e.g. configurability, upgradability);
- the (generic) customer has become a much more influential stakeholder;
- more attention has to be given to integrating with other systems.

These coupled with the reduced life cycles typical of software products, means that a broader stakeholder discourse must occur, often under strict time-pressure. This thesis will provide a means for such stakeholder discourse in the form of an assessment method operationalised for two specific family-relevant flexibilities: interoperability and extensibility.

3.4.5 Potential solutions

The aims at codifying current practice and providing tools, and discussion forums discussed in section 3.4.3 are all positive movements towards addressing the challenges of developing software-system families. As in the general case of product families (and as has been recognised by the SEI in the case of software-system families), using the architecture as a means of co-ordinating stakeholder co-operation is a fruitful option. This is especially so in the case of software, whose abstract nature and inherent flexibility contributes to its overall complexity. This complexity makes stakeholder communication more challenging, and the role of a tool to manage complexity even more important.

3.5 Information-system families

3.5.1 What?

Information systems are a special case of the general software system discussed earlier in section 3.4.1.

Definition:
Information systems: "*Computer-based systems intended to provide recording and supporting services for organisational operation and management*". [Verrijn-Stuart, 1989]

Information-system families are, consequently, a special case of software-system family, where the software system in question is an information system. Performing the appropriate refinement to the previous definition for software-system families results in :

Definition:
*an **information-system family** is a set of information systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or customer; built from a common set of core assets*.

Some general characteristics of information systems have already been presented in Chapter 1. These systems exhibit all the characteristics of software systems discussed above, and emphasise the need for flexibility, long-term business relationships and interoperability.

Another key characteristic of information-system development and deployment is that until relatively recently these systems were typically custom developments. This means that each system was developed as a *single* project for a *single*-customer, often by *in-house* information system (IS) departments. From this perspective the concept of multi-customer, product-based

development associated with product families is new; not only for the user community but also for the producer community and their attitudes to information-system architecture and development. This adds to the difficulties of applying product-family concepts in these project-oriented environments [Bosch, 2000].

Information Box 3-B: What is so special about software families?

Opportunity

Software product families are becoming the norm rather than the exception. This is especially so when we see software being incorporated in existing (e.g. automobiles, again) families as a component; the family experience will tend to be applied (or reused) to the software parts. Even in the situation where the software is so significant that it is managed as a (marketable) product in its own right, family techniques are appropriate as most companies producing such (software) products are concerned with customisation and multi-release life-cycle management. Further justification for the suitability of a family approach to software development is the amount of attention paid to legacy support in any new release projects and the significant effort invested in software maintenance. The central topic of this thesis – architecture – is also closely associated with the notion of family; authors like Jazayeri, *et al*. [2000] see families and architecture as mutually enforcing concepts, the demands of the family being necessary to warrant the architecting effort. The issue with software families is not the acceptance that they are the right way to go, the issue is in *how* family concepts can be implemented in the software context.

Problem

There are two main categories of reasons why this is difficult in the software context:

- Subjective – software customers, managers and developers read the word as "SOFTware"; they are used to operating in a state of constant flux where everything moves quickly, decisions are made, revisited and reversed in days, where feature-creep, long integration cycles and never-ending commissioning projects are regarded as normal. The flexibility of software, means that people have got used to postponing firm decisions, because it (the software) can be "easily" changed. Those developing families of non-software systems, where flexibility is not so readily available and significant risk and investment is associated with volume manufacturing and distribution, have acquired more discipline and are used to stricter operational management and strategic (long term) planning.
- Objective – the abstract and flexible nature of software (its strength) means that much is possible, and there is always a market for these possibilities. The amount of cost-effective variety, and flexibility enabled by software means that there is more market variety to manage. This coupled with the pace of change in the industry means that there is a lot to manage in an environment of risk and uncertainty.

Software is young and is changing (or trying to) from a craft to an industrial activity in a generation. There was two hundred years of rifle manufacture before standard interfaces made parts interchangeable; Ford's car industry removed variety from the equation in order to optimise. Software is different from hardware, but for the successful practices at the heart of family development the issue is scale: bigger, faster, more, in less time. Scale, however, matters.

Some key family characteristics

Much modern software is produced with many of the characteristics we see from product family thinking. The key differences between ongoing single-product development, upgrade, extension and product family development are [Bass, *et al*., 1998]:

- The production of a related set of products – multiple products have individual life-cycles within the product family; but each product's evolution must be managed from the perspective of the family as a whole;
- Their production from a core asset base, following a specified production plan

In software, the realisation that the result of the current development project will be built on, will be used for years to come, will be subject to change and demands for more change, is the "great leap forward" in starting and maintaining a family approach. All too often projects never look beyond their immediate, short-term scope to keep the (single) customer happy – today. The key attribute of family-oriented development is the concern now, for future generations and for other family members.

The principle of core assets as a basis of deriving products is similar to reuse of standard parts in conventional family practice. Proposals for organising development organisations around the principle of core-asset reuse have been provided by Jacobson, *et al*. [1997] and Bass, *et al*. [1998], and activities such as Domain Engineering for identifying commonality and variety are advocated as part of the practice framework provided by Clements and Northrop [1999]. Many of the important system qualities underlying families such as configurability, upgradeability, and interoperability are motivated by the need to reuse assets.

Interfacing between components is one of the most important aspects of family management. It dictates how well the whole will work together and how change is accommodated and propagated throughout the family. Some technical approaches supporting this part of asset management include API (supporting extension), parameterisation (supporting variety), version independent interfaces (supporting evolution, see e.g. [Chappell, 1996]), and technologies such as XML (supporting interoperability).

The important point is not the implementations used to provide ever more de-coupled interfaces. You cannot say that a system is not part of a family if it does not have version-independent interfaces. This undoubtedly helps, but of more importance is that interfaces are explicitly managed, there is a clear policy and practice when dealing with component interfacing and that the interfacing policy supports the flexibility and reuse goals of the business. It is entirely practical (i.e. efficient design) that those parts of the system experiencing most change and variety need version-independent or "open" interfacing, but for those stable parts of the system this approach may not be cost effective.

Information-system families

Because of their high levels of customisation, their connectedness (as part of connected business chains), their longevity in a changing environment, and their distance from the volume experience of non-software or embedded-software practice; information systems are particularly challenging to product-family techniques. A distinct advantage, however, is that while other software consumers may not know that a system comes from a common-asset base as part of a family [Bass, *et al*. 1998], the information-system customer is concerned with long-term issues and may appreciate that there is a family approach. This appreciation can be leveraged in product-family planning and marketing in trying to achieve shared producer-customer objectives.

3.5.2 Why?

Information systems are a fact of contemporary business life; and the reasons to pursue a product-family approach to their development is motivated by the fact that many prime incentives for product families are characteristics of the current information-systems business *viz*.:

- intense buyer-driven competition,
- large demands for customisation[4]
- an incentive to exploit reuse (currently fuelled by a skills shortage)

The intention is to apply the benefits of product-family based competition (as proven in other domains) to the information-system domains. Of particular importance to the information-system industry is the continual evolution of products over time – management of information integrity, and functional compatibility across versions and generations in an industry plagued[5] with ever-shortening release cycles is a real problem. It is intended that the lessons on stable, generic interfaces associated with product families can be applied to information systems.

3.5.3 How?

As with product families in general, there is a scarcity of generally available data on how the family approach is successfully implemented. It is further exacerbated in this case – as information systems are only recently emerging from the single-product paradigm. In the absence of evidence to the contrary, it is probable that most information-system family development is (as with the majority of industry) on an *ad hoc* basis with companies educating themselves. A summary of the general state of practice regarding practical techniques for software-system families is provided previously in section 3.4.3.

3.5.4 Challenges

Information-system family development inherits the general challenges typical of all (software-system) family development (see sections 3.3.4 and 3.4.4). In particular, the fact that information systems are (see also Information Box 3-B):

- subject to ongoing modifications throughout their life cycles;
- increasingly expected to operate in close co-ordination with third-party systems;

means that system qualities (the "ilities") have become very important business enablers. As indicated in Chapter 2, system qualities such as: extensibility, interoperability, reusability, scalability, and portability are all key competitive aspects of the information-system family business. These aspects have, however, been traditionally under-represented in information-

[4] This demand is at least in-part due to a widely held belief that software is infinitely flexible at no extra cost – co-incidentally an "axiom" that is not actively resisted by those selling and marketing software (source: discussion with R. Kusters, TU, Eindhoven, June '97)

[5] Multiple versions released in rapid succession are a blessing or a curse depending on your viewpoint, but are generally warranted by the fact that incorrect promises were made for the previous version or the correct promises have not been met.

system development. There is work to be done in raising awareness of these issues for family stakeholders and then propagating these concerns through design and development.

3.5.5 Potential solutions

Again architecture is the main solution concept promoted in this thesis to help the family stakeholders address the important system qualities that determine the families success. Information-system families have an increased business orientation for both customers and producers (they directly support business processes), in particular their multi-user and multi-systems usage context means that they have a broad scope. This breadth means that a wide range of issues must be addressed and therefore the number and range of stakeholders increases. The position of architecture as a common framework where these multiple issues must be addressed makes it a natural place to address the associated challenges.

The following section provides a broad treatment of software architecture, with special attention given to its role in communication, and managing complexity. The central role of architecture in supporting the business strategy of the family is the prime motivation for its potential to address the challenges in information-system family development – how architecture fulfils this role is explored therein.

3.6 Software architectures

Architecture is the central theme of this research. Initially software architecture will be introduced, in line with its treatment in literature and methods, in a general way with sporadic reference to family concerns. After an overview of terminology, the term "architecture" will be defined in the context of this research. Hereafter the rationale for concentrating on architecture will be made, and increasing emphasis on family issues will be used to show that architecture is a critical concept for the development of information-system families. In succeeding sections the use of the architecture in addressing the commercial aspects of family business mentioned previously, and the accompanying challenges to development, will be explored. Finally a set of architectural challenges and opportunities for meeting them will be presented.

The prime aim of this section is to provide a comprehensive picture of the essence of software architecture and its application in this research. It is, however, not the intention to provide an in-depth analysis of all aspects[6] of architecture representation, or practice; that has been the subject of complete books, many of which are referenced herein.

[6] Readers interested in a comprehensive discussion on the history of architecture and its classical roots are referred to [Zwegers, 1998] Chapter 2.

3.6.1 What?

Following is a comprehensive overview of the terminology and research findings concerned with the definition of software architecture. A synthesis of the information is found in section 3.6.1.4, and the position of software-family architecture presented in section 3.6.1.5.

3.6.1.1 Overview of definitions

Architecture is a widely used (perhaps abused) term in product-development literature, and it has been applied to almost all aspects of design. Software architecture, in particular, has been the subject of much recent academic and industrial interest. While there is yet no commonly agreed definition, there are many variants (more families!). Following is a chronology of recent contributions:

1. Perry and Wolf, [1992] – talk of set of design elements that have a particular form. These elements can be categorised as process, data, and connection;
2. Garlan and Shaw, [1993] – regard (software) architecture as a level of design related to "*...specifying the overall system structure...Structural issues include gross organisation and global control structure; protocols for communication, synchronisation and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among alternatives*.";
3. Garlan and Perry, [1995] – "*The structure of the components of a program or system, their interrelationships, and principles and guidelines governing their design and maintenance over time*.";
4. Soni, *et al.*, [1995] – empirically define the following four (software) architectural perspectives:
   - the **conceptual** architecture describes the system in terms of its major design elements and the relationships between them;
   - the **module interconnection** architecture encompasses two orthogonal structures based on functional decomposition and layers;
   - the **execution** architecture describes the dynamic structure of a system;
   - the **code** architecture describes how the code is organised in files, binaries, and libraries;
5. Kruchten, [1995] – proposes the 4+1 model, comprising 4 views with scenarios to illustrate them:
   - the **logical** view describes the design's object model or entity relationship model;
   - the **process** view describes the design's concurrency and synchronisation aspects;
   - the **physical** view describes the mapping of the software onto the hardware;
   - and the **development** view describes the software's static organisation in its development environment;
   - The **use-case** view is used to provide illustration as to how the individual view elements co-operate;
6. Van der Linden and Müller, [1995] – introduce a 3-D architectural model as part of their Building Block (BB) development method. The dimensions are:
   - **structure** – describes the BBs, layers and their relationships;

- **aspect** – describes the functionality, and is orthogonal to the structure;
- **behaviour** – describes the design of concurrent elements and their synchronisation;

7. Garlan, [1995] – presents a distillation of definitions and viewpoints presented at the First International Workshop On Architectures For Software Systems. The views are:

- **structural models** hold that software architecture is composed of components and their connections plus some other aspects, (e.g. configuration, style, constraints, properties, rationale, requirements);
- **framework models** are similar to the structural view but they emphasise the coherent structure of the whole system rather than component details. Examples of these models are domain-specific software architectures;
- **dynamic models** concentrate on the behavioural aspects of systems; which may refer to changes in system configurations, interaction dynamics, or data transformations during processing;
- **process models** focus on the construction of the architecture itself from a process script or programme;

8. Ellis*, et al.*, [1997] and the Architecture Planning Group, under the auspices of the IEEE, – argue that the level of abstraction of architecture must be emphasised[7] and that its concern with the context of the system promoted. They define architecture as: "*... the highest-level concept of a system in its environment*" and further they separate this from the *representation* of the architecture: "*an architectural description is a model – document, product, or other artefact – to communicate and record a system's architecture……*(it) *conveys a set of views each of which depicts the system by describing domain concerns*";
9. Bass*, et al.*, [1998] – define architecture as:
   "*…the structure or structures of the system, which comprise software components, the externally visible behaviour of those components and the relationships among them*.". They emphasise the abstractive nature of architecture and its concentration on the external behaviour of components and interaction, relationships with others. The emphasis on behaviour means that simple box-and-line drawings are not complete architectural stories – supplementary specification of the component interfaces is essential to architecture description.

3.6.1.2 Analysis

The above definitions represent a spectrum of opinions in the architecture research community as to where the emphasis in architecture should lie (to summarise Garlan, [1995]): its constituent parts, the whole entity, its behaviour when built, or the process of building it. It is fair to conclude from all the definitions, that the essence of product architecture is capturing the structures of a system, and the relationships among the elements both within and between structures [Soni, *et al.*, 1995]. The same source, and others (e.g. [Clements, 1994]) also stress that these structures are not the same, and attempts to merge them are fundamentally flawed

[7] This follows Shaw's advice at IWASS (see [Garlan, 1995]) "let's not dilute the term 'architecture' by applying it to everything in sight"

as the separate structures are optimised to meet different development criteria. This realisation of the folly of a "Grand Unified Representation" – THE architecture model – was also seen in the information-systems community, when Zachman [1987] realised that each actor (stakeholder) in the development process needs his own (set of) representation(s).

One of the seminal works in the area [Perry and Wolf, 1992] has formulated a concise definition of this "structures of related components" theme, and provided an important addition, by defining (software) architecture as:

(software) architecture = {elements, forms, rationale}

This definition captures the essence of architecture:

- *elements* – the different components in the system, the authors identify three types for software systems: processing, data, connecting;
- *forms* – the various different structures or views on (a selection of) the elements;
- *rationale* – the design reasoning including quality attribute aspects.

The importance of rationale is crucial, as it extends architecture beyond a set of diagrams, and stresses the importance of continuity, analysis, and learning by providing for design rationale and constraints to be recorded. The notion of rationale is also important for its affect on later authors, especially those who use the system requirements as a general basis for expressing the rationale [Gacek, *et al.,* 1995] and in particular those who regard use-case based requirements as a tool for expressing and analysing architectures ([Jacobson, *et al.,* 1992], [Kruchten, 1995] and [Kazman*, et al.*, 1996]).

The work of Gacek at USC [Gacek, *et al.*, 1995] is built on the original formulation of Perry and Wolf, and maintains that a ***software-system architecture*** *comprises:*

- *a collection of software and system components, connections and constraints;*
- *a collection of system stakeholders' need statements;*
- *a rationale which demonstrates that the components, connections, and constraints define a system, that if implemented, would satisfy the collection of system stakeholders' need statements.*

The authors further imply the need for architecture *representation schemes* to support reasoning about the architecture's ability to support stakeholder needs. They advocate alternate views of the architecture including at least:

- a behavioural and operational view;
- a static topological view;
- a data-flow view;

and a notation capable of capturing other views, and including attributes which support reasoning about stakeholder-critical properties e.g. cost, performance, portability.

This broad definition of architecture, with reference to requirements and rationale, shall be interpreted for the *spirit* of the sentiment rather than the *literal* content. In this research the system requirements is regarded as being a separate artefact from the system architecture; but the two are closely related, developed simultaneously, and should support each other, the support being explicitly recorded in the "rationale" aspect of the architecture.

Some notable aspects of Gacek's definition are:

- the addition of "constraints" to the view-contents – Ellis*, et al.*, [1997] have described these as laws that the system must observe, applying to components and connections, and generally governing: system behaviour and properties; architectural style and protocol; and natural laws.
- its strong emphasis on stakeholders; a pragmatic attempt to place people centre-stage in architecture.
- recognition (along with others e.g. [Hammer, 1996]) that multiple architecture representations (views) exist; and especially that behavioural and dynamic aspects are architecturally significant.
- recognition that architecture and requirements are very closely related; and calling for verification that the architecture meets the requirements.

This comprehensive definition of software-system architecture with its emphasis on explicit rationale to ensure the satisfaction of stakeholder needs, will be adopted in this thesis as a *reference framework* in which to position subsequent research activities. The importance of rationale in providing a means to communicate architecture quality (with respect to requirements) is especially relevant to this research and the assessment method is a means to extract, record and communicate this rationale.

Following, a summary of some key terms common in architectural parlance is presented to clarify their position in this research framework.

3.6.1.3 Architectural styles, reference models, and reference architectures

There are three other concepts that are common in the architecture literature, and will be briefly defined here – this section is adapted from [Bass, *et al.*, 1998] pp25-26:

Architectural style – is a description of component types and a pattern for their runtime control and/or data transfer (it can also include topological interconnection). A style can be regarded as a set of constraints on the architectural components and their interactions. Client-server is a common architectural style – it implies that two component types exist and that their co-ordination is defined in terms of the protocol the server uses in communicating with the clients. The style is not an architecture – the clients are not identified, nor are the (non-protocol) aspects of the client or server functionality specified. Style conveys a useful image of the system and imposes important constraints.

Reference model *– is a division of functionality together with data flow between the pieces*. It provides a standard composition of a known problem into parts (sub-problems) that co-operatively solves the problem. Reference models are characteristic of mature problem-solution domains, e.g. reference models for compilers or database management systems.

Reference architecture – is a reference model mapped onto software components that will co-operatively implement the functionality defined in the reference models and the data flows between the components. Whereas a reference model divides the functionality, a reference architecture is a mapping (not necessarily 1:1) of that functionality onto a (generalised) system decomposition.

The above concepts are not system architectures, but they are useful steps towards an architecture (see Figure 3.2), and represent early design decisions that shape the architecture. The important point is that reference models concentrate on the problem domain, while reference architectures address the solution domain, in a general (company unspecific) way.

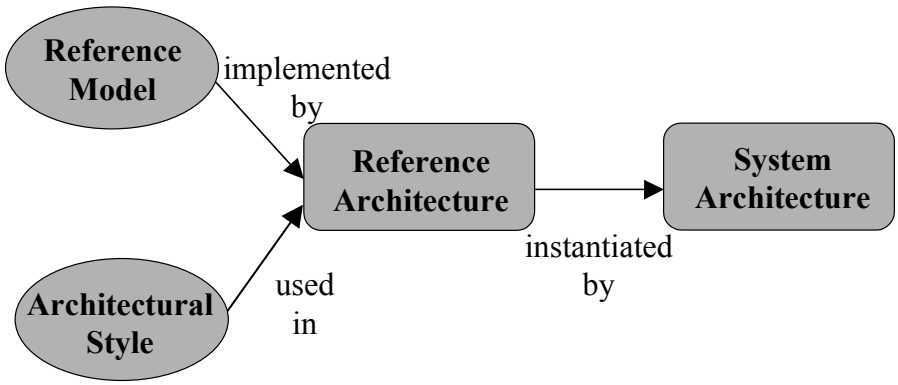


Figure 3.2: Relationship between architectural style, reference architecture and software architecture (adapted from [Bass, *et al*., 1998])

3.6.1.4 Synthesis

Taking Gacek's broad definition as a reference frame some synthesis can be woven from the various contributions so far – and some new ones. These tenets represent the approach to architecture taken in this thesis:

- Architecture is concerned with high-level internal-structure***s*** of a system, addressing both *static* and *dynamic* aspects of the system.

(software) architecture =Σ view{style(elements, constraints, forms), rationale}

- **view** – an architecture is experienced mainly through representations of selected views on architectural issues; each view is a model (document, diagram, simulation) of a high-level structur***e*** of a system's components, that abstracts away details of the elements' internal

implementation, algorithm, and data representation; but shows the externally visible (to other elements) properties of these components and their relationships. (*adapted* from [Bass, *et al.*, 1998]).

The candidate views (4+1) proposed by Kruchten [1995] – logical, process, development, deployment, and use-case – are a representative set of useful views, provide use-cases for communication and to illustrate dynamic aspects, are supported by the UML [Rumbaugh, *et al.*, 1999] standard, and will be adopted as good architectural practice by this thesis.

- **style** – an architecture view typically follows a particular *style* [Shaw and Garlan, 1996] that provides a template for the structure (form), content (elements) and may also provide rules (constraints) governing element-relationships. Styles emphasise different aspects of architectural-level concerns, and this is reflected in the notation and elements covered by the form e.g. client-server, OO, entity-relationship.
- **Σ** – multiple *views* of architecture can exist, the representations and their contents are completely dependent on what the stakeholders using the *views* want to study; it is therefore useful to regard architecture as having a multi-disciplinary aspect [Maier, 1996]. Multiple views are necessitated primarily by the fact that different aspects are treated separately by stakeholders (e.g. logical and physical), and that different views are appropriate at different phases of the life cycle (e.g. logical – early, and physical – late).
- **elements –** based on [Perry and Wolf, 1992] and [Bass, *et al.*, 1998]; elements are characterised as those data-storage, data-processing (or both) entities and the relationships between them. Elements are presented in the various forms – and guidelines on element typology are generally provided by the choice of *style*. Elements may appear in multiple *forms*, and elements in one form may be related to those in another. It is important that the externally visible properties of the elements are provided in appropriate forms.
- **constraints –** these are rules (laws) governing the static and dynamic relationship between the elements in the various views. In many cases *styles* provide constraints on the possible relationships between elements. Constraints may also be used to provide requirements regarding selection of technology, or other organisational and process rules that must be followed during development – in this sense it captures the non-requirements [Bass, *et al.*, 1998] influences on architecture.
- **forms –** is the notation used to represent the view, this is heavily influenced by the view and the style, e.g. UML class diagram
- **rationale** – an explanation as to what the architecture representation is actually showing and why, is very important to bring the forms beyond "boxes and lines" (see illuminating and entertaining example of this in [Bass, *et al.*, 1998] pp21-23). Gacek, *et al.*, [1995] argue that the rationale is very important particularly when it provides requirements-based explanations of design choices.

General Notes:

- All systems have an architecture[8] (e.g. the human body, a termite colony), however, it may not be represented, nor understood by many, in fact there may not be an architect!
- Requirements are not part of the architecture – they are important in providing input and context for the architecture, and in helping communicate the architecture to stakeholders.
- Bass, *et al*.'s [1998] **definition** of ***software architecture*** as:
  "*…the structure or structures of the system, which comprise software components, the externally visible behaviour of those components and the relationships among them*."
  is adopted in this research as a useful description of the key attributes of architectural views.

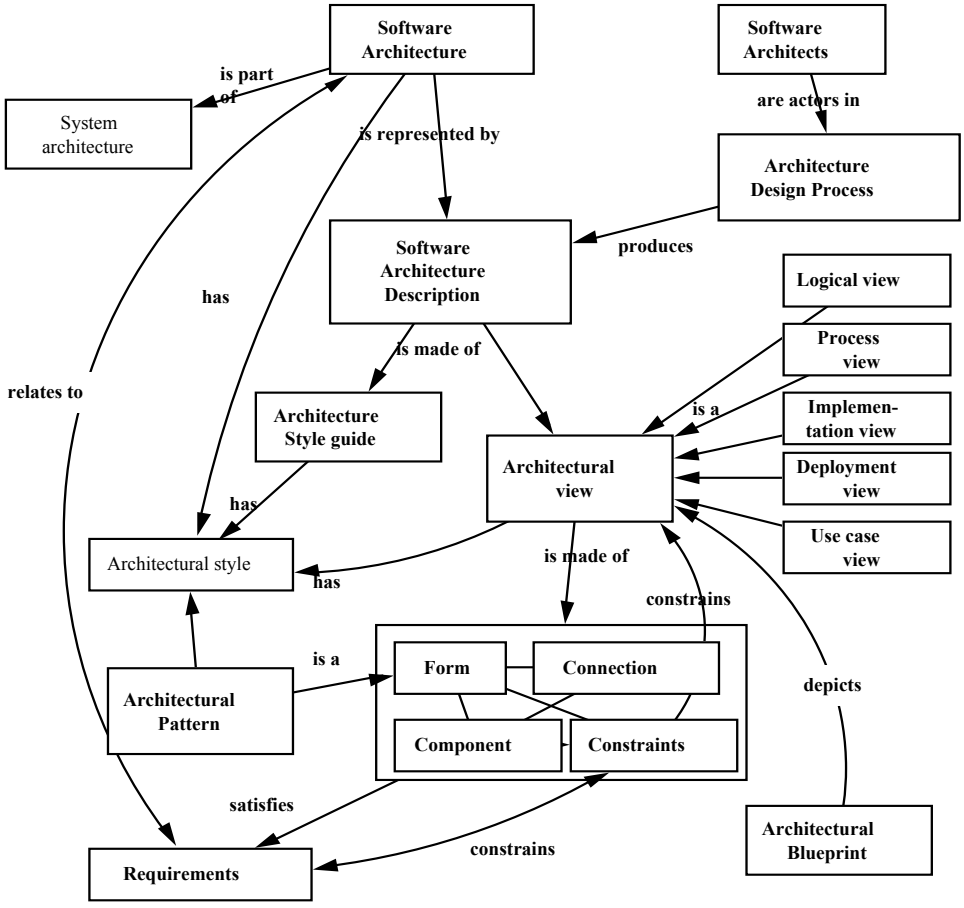


Figure 3.3: Architecture concepts and relationships (© Rational Software Corp. 1999)

The above figure is reproduced from a popular CASE vendor's infoware and graphically illustrates the relationship between the terms discussed earlier (and some others). It gives an impression of the number and richness of architectural concepts and their (complicated)

[8] DeMarco makes this point eloquently when describing Mumbai's (Bombay's) "box-wallah" home-cooked lunch delivery service – see [DeMarco, 1995].

interactions. It is left as an exercise for the reader to determine if this is an architecture of architecture !

3.6.1.5 Family architectures

The discussion on architecture thus far has been neutral with respect to single-products versus product families. It is now appropriate to place architecture in a family context. A recurring theme in much of the literature is that architectures and families go hand-in-hand. In particular, it is the reuse of the architecture (by the producer) across family members (sold to customers) that is the main motivation, and justification, for investing in architectural design.

Families are characterised (see section 3.3) as comprising generic reusable components some of which are customised to provide the necessary variety demanded by family customers, and others which are standardised across family members and which provide scale economies through repeated direct reuse. Family architecture is no more (and no less) than the collected structures of these generic family modules, in relation to the previous definition on software architecture:

Definition:
Information-system family architecture – is *a collection of models, each reflecting a high-level structure of the information-system family's generic components, showing the externally visible (to other components) properties of these components and their relationships. This family architecture is used to derive particular system architectures for individual family members.*

Family architecture is, therefore, a special case of architecture, recognising that:

- the elements (components, connections) are generic family components and connections that provide variety and reuse possibilities across individual members;
- the nature of the element descriptions is typically more abstract (see e.g. [Kazman, *et al*., 2000]) than that of the single-system approach in recognition of the need for more customer-specific and time-specific independence;
- the family nature of the architecture is also reflected in the architectural *constraints* guiding the relationships between components which must carry information on how the generic elements can be combined or instantiated to form the individual family member systems (see Figure 3.4).

In many other respects the nature of the information-system family architecture is similar to that of any software system, allowing classical and emerging architecture techniques to be reused in the family domain.

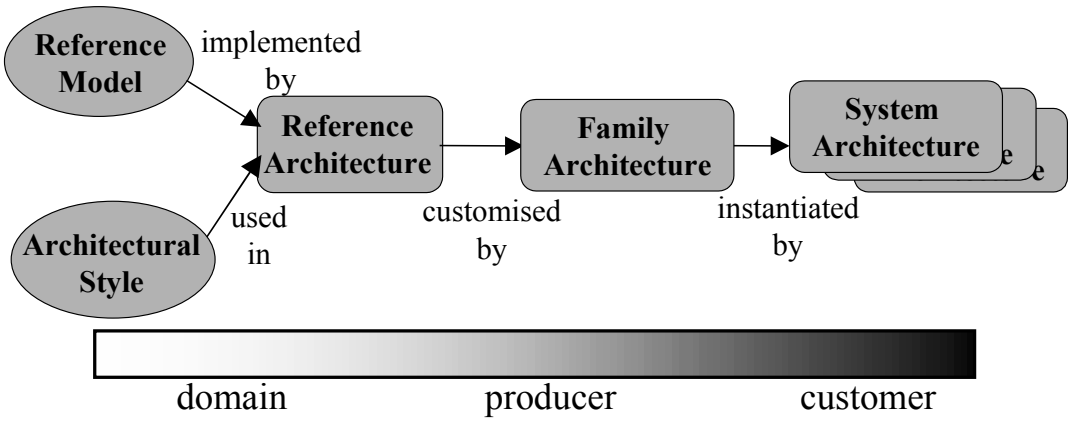


Figure 3.4: Family architectures are used to derive individual system architectures for family members

In relation to the, typically industry-wide (see [Bass, *et al.,* 1998] pp. 376), reference architectures discussed earlier in 3.6.1.3; the family architecture is specific to the producer and reflects how the producing organisation customises the domain reference architecture to suit its own business strategy and organisation. The family architecture details the reference architecture further for use within the producing organisation. As shown in Figure 3.4, it represents an intermediary step between the reference architecture and the individual system architecture depicted earlier in Figure 3.2.

The family architecture is where the business-specific concerns of the family-producer are introduced and where details of family members and their interrelationships are described. Of particular importance for this research is the fact that the reference architecture provides a decomposition from problem to software; but this is typically function driven, and does not address those system qualities needed to actually manage and market the software for the producer over time. The family architecture enriches the reference architecture by addressing the more strategic business-driven system qualities important for the family.

Additionally, the company-specific nature of family architectures allows multiple producers to derive multiple family architectures from a single reference architecture.

While the definition used above follows logically from that definition applicable to single-product architectures; the devil is in the details. The distinguishing feature of family architectures is not the textual definition – but rather the generic, multi-product, future-oriented nature of the problem, and hence the architectural elements. The software modules and their interfaces captured in the family (development, see 4+1 views) architecture must account for the variability and reuse across the multiple individual systems anticipated. For example, to address the need for uncoupled component evolution, the family architecture may provide a specification of "version-independent interfaces" guaranteeing backward compatibility with pre-existing components. The move from single-product, to family architectures is an issue of complexity and scale – there are more systems with more variety,

more dependencies, over longer time periods. The fact that it is complexity and scale of known issues does not diminish the effort involved – in families, scale matters.

The family architecture defines what is fixed for all members of the family and what is variable. This may result in sub-optimality in some member systems, but this is compensated by the reuse-based savings in risk and development effort [Bass, *et al*., 1998]. This is similar to house design, which is seldom from scratch but based on minor (typically non-structural) modifications to a standard design. The consequence for those used to buying (and selling) under the single-system approach is that now extra compromises "for the sake of the family"[9] are introduced, and this represents a change in mind-set where *thorough* negotiation with customers regarding the importance of requirements is necessary.

The key difference between family architectures and single-system architectures is that business decisions drive the former, while technical decisions drive the latter.

3.6.2 Why?

> *"If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process"*
>
> – Barry Boehm, 1995

Architecture is an important medium for managing risk in all industrial software development. Literature on architecture is extensive and increasing, and this section describes a collection of the most relevant applications of architecture in industry. As stated previously, most attention in literature has been on single-system architectures; the architectural issues are still present, and indeed exaggerated, in the family situation; and are therefore relevant.

1. **Complexity management** – architecture is concerned with high-level abstractions of systems, and this is useful for reducing the complexity that must be dealt with at any one time [Simon, 1981]. One of software architecture's early pioneers, Brooks [1995] has identified complexity as an essential property of software and has argued for architectures role for helping development cope with this. Zachman [1987] has identified architectures role in controlling complex inter-component integration during information-system implementation.

2. **Communication amongst stakeholders** – architecture representations are common high-level abstractions of a system and can be used as a forum for shared understanding, consensus and communication [Bass*, et al.*, 1998]; architecture is especially useful in

[9] Some customers may indeed *initially* perceive family development as a mafia-style negotiation process.

separating concerns and so facilitating decision-making [Dolan, *et al.*, 1998]. Multiple structures in the architecture emphasise different aspects of the system. In particular (see [Bennett, 1997]):

- build-*time* structures are important during the development and maintenance phases of the system's life cycle – an example structure will emphasise the calling dependency between software modules in the development environment (the module interconnection architecture [Soni, *et al*., 1995]);
- *run-time* structures are concerned with the running system in its operating environment – example structures will emphasise the synchronisation and ordering of the processes in the running system; or the organisation of the various hardware and software resources used to deploy the system (the physical-deployment architecture [Kruchten, 1995]);

Architectures are concerned with structure, and are a source of order [Zwegers, 1998] in the sometimes chaotic world of software development, an additional aid, therefore, to stakeholders in resolving conflicts.

3. **Indicates most vital system elements and decisions** – The architecture contains the earliest and most important system decisions. These decisions guide development and have a dominating influence[10] on the systems qualities, according to Clements, *et al.*, [1995]:
   - the modifiability of the system, for instance, depends on the degree of modularity prescribed in the architecture;
   - the reusability of common family-features depends on the coupling[11] (integration) between components;
   - system performance is heavily influenced by the volume and complexity of inter-component communication across the various interfaces;
   - interfacing also dictates how open the system is to integration with other systems in the usage environment (interoperability), and how amenable it is to (re)using off-the-shelf components.

   The above qualities have earlier been indicated as of critical importance to information-system families. The architecture is also the earliest point in the system's life cycle when the system can be analysed [Bass, *et al.*, 1998] to answer what-if questions. It is, therefore, an excellent opportunity to assess how well early design is aligned with strategic objectives.

[10] It is important to understand however, that while a good architecture is *necessary* to ensure quality; it is not *sufficient* as inferior downstream design. Implementation can always compromise architectural design [Bass, *et.al*, 1998].

[11] Coupling (amount a single module "knows" about other modules, [Berard, 1995]) is a more technical term used when describing the strength of connection or dependence among system components. Low coupling facilitates module reuse and replacement.

4. **Determines system development and evolution**

- the architecture describes the components of the system and their relationship with other components – this enables independent, parallel development and provides a framework for system integration and modification.
- Its attention to interfaces largely determines how the system can be sold, maintained and re-configured.
- The architecture is a record of the mapping of requirements to solution modules, and the selection of components and communication *interfaces*. In product-family development these decisions will be *reused* across different product lines and thus their consequences are magnified. Architecture is, thus, an embodiment of corporate memory and competitive strategy,
- Finally the (4+1 development view of) architecture and the software-development-organisation structure are typically mirror images [Clements*, et al.*, 1995]. This means that the architectural decisions are institutionalised in, for example, team structures, work assignments, integration procedures. This is necessary to develop clusters of competence in the organisation, and means that architecture facilitates (and constrains) organisation learning. It also means that any changes to the architecture must be accompanied by appropriate organisational restructuring[12].

  Meyer and Lopez [1995] summarised these ideas: *"Architectures are both a basis for product innovation and a constraint on the variety of product versions that can be offered."*

5. **Means to gain strategic benefit** – The architecture is a manifestation of the technical strategy underlying the firm's business strategy for the system (family). An increasingly interconnected world architecture can enable non-core components to be bought-in commercially (so-called COTS components) allowing the firm to concentrate on its real competitive edge. Conversely, the incorporation of standard interfaces in the architecture may allow the firm's system to be incorporated as an essential part of third-party systems, thus establishing extra market and perhaps a monopoly position for the firm [Morris and Ferguson, 1993].

6. **Supports product-family development** – The most significant aspect of architectures (for this research) is that it is a natural supporter of family-based competition. This is clear from the preceding discussion, where many system qualities important to families were seen to be addressed by architecture (modifiability, interoperability, complexity management). System architects use the architecture to define which architectural aspects are common for all members of the family and which are variable. This enables family-

[12] This has also been termed Conway's Law: "Organisations which design systems are constrained to produce systems which are copies of the communication structures of the organisation." (Datamation 14, 4 April 1968).

critical commercial decisions on reusability (to economise) and flexibility (to provide variety) to be translated to the technical domain and to be propagated to the development organisation. Architecture is a key asset in operationalising the family-strategy.

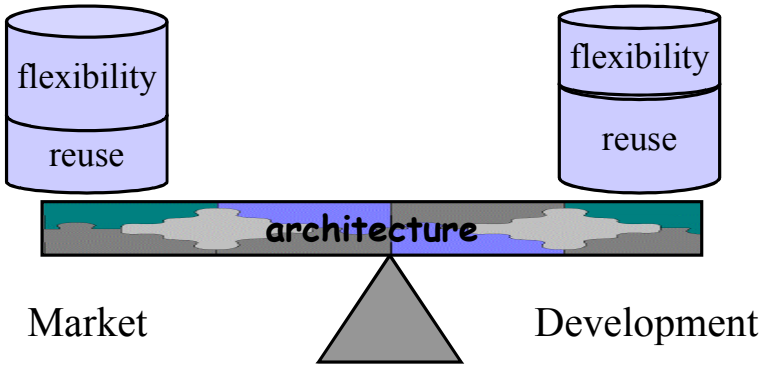


Figure 3.5: architecture balances market and development demands for flexibility and reuse

The product-family architecture plays a dominant role in several key aspects of the family, and is a crucial ingredient for family success. Figure 3.5 graphically summarises the main forces at work in software-system family development, and illustrates that the architecture operates in a market and development influenced "see-saw", and acts as a focal point around which the family is balanced. The architecture, therefore, deserves the close attention of all those concerned with the success of the family – the stakeholders.

3.6.3 How?

The process of creating and building architectures has been termed "architecting" by Rechtin [1991]. Architecting is a complex, human-centred activity (see previously). It is essentially a balancing-act, where the architect tries to account for all of the various forces that must be addressed by the product (see Figure 3.6); and the resultant architecture represents decisions made in providing an optimal match between the:

- demands of the market;
- capability of the technical solutions;
- characteristics, constraints, and resources of the organisation.

Due to the:

- unbounded nature of systems (most systems are contained in, and interact with, other systems);
- large number of complex interactions among system elements and requirements;
- absence of detailed information in the early stages of development;

some authors have highlighted the limitations of the traditional scientific method (with an emphasis on precise measurement, independent variables, and neat problem boundaries) to architecting [Rechtin, 1991]. In Rechtin's view, the primary tool of the effective architect is common (contextual) sense, which is often articulated in heuristics. These heuristics are built

up over many years of personal and professional experience, some examples are provided in Rechtin's book.

This view of architecting as a person-driven, craftsman-type activity which has more in common with art than science is echoed by Bennet [1997] when he says that although there is lots of talk about architecture recently there is little general consensus of what it really involves and doesn't involve. Clements and Northrop [1996] recognise that although the roots of modern architecture were laid by Dijkstra in 1968, since then practice has been leading study and currently study is attempting to codify all the disparate activities carried out by designers (some of which may not be strictly architectural). Bass, *et al.* [1998] have presented some basic architectural techniques (e.g. abstraction, resource sharing, decomposition) which they term *unit operators*, and have organised them and their interactions as a means of codifying architectural practice. Hofmeister, *et al*., [2000] (from Siemens research) have codified best architecture practice at Siemens and describe (with extensive case material) how to apply *global analysis* with their previously defined 4 views (see [Soni, *et al*., 1995]) in developing real industrial architectures. The popular pattern work by Gamma, *et al.*, [1995] also codifies good design (rather than architectural) practice in terms of proven problem-solution pairs which aims to reuse design knowledge both within and across application domains. Codification of practice is welcome, and (as in other fields) provides concrete evidence of the emergence of a professional body – the body is still very much in its infancy however.

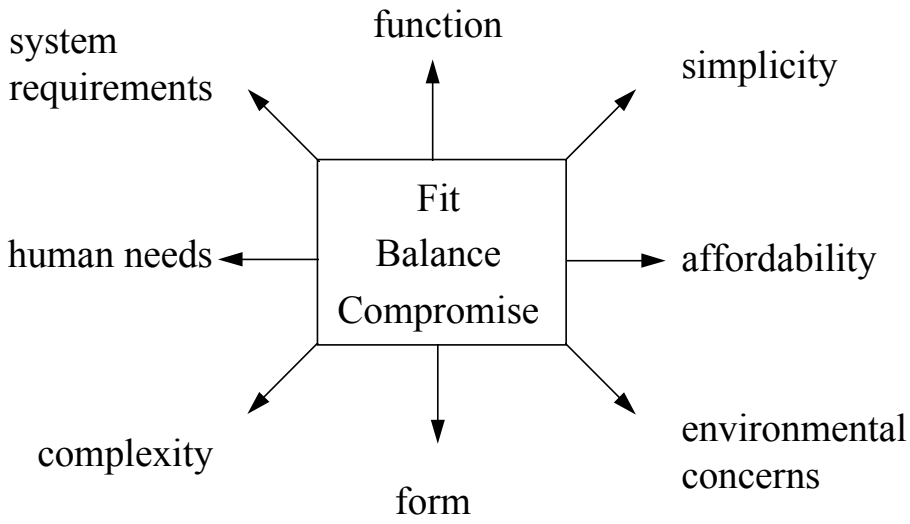


Figure 3.6: Some tensions in systems architecting, from [Rechtin, 1991]

Rechtin [1991] provides some insight into the primary responsibilities of the architect, contrasting architecting with engineering as an illustration; summarising:

- architecting is function-based while engineering is form-based
- architecting reduces complexity, ambiguity, and uncertainty to workable concepts, while engineering makes feasible concepts workable;

- an architect can be regarded as working *for* the client (an idea originally associated with [Brooks, 1995]) *with* the builder, whereas engineering can be regarded as working *with* an architect *for* the builder;
- architects tend to concentrate on concepts, synthesis, top-level specifications, technical and non-technical interfaces, project success; while engineers tend to concentrate on defined sub-system interfaces, analysis and performance to specification.

Apart from the technical issues at the centre of architecting there is a large socio-political aspect inherent in the role, because of its:

- strategic nature;
- role as a forum for decision making (and hence compromise);
- direct impact on the organisation and its members (e.g. make-buy debates etc.).

This is steadily being appreciated by the industrial and academic architecture communities, where there is increasing interest and debate on so-called "people" issues (see e.g. [Cockburn, 1998], [Obbink*, et al.*, 1998]).

Summarising: there is no standard, accepted architecture practice. The process is driven by experience-based heuristics and the dependence on the individual's professionalism. There are however some moves towards integrating architecting in the mainstream of software engineering. Codification of practice is emerging from research-industry partnerships. Some initial steps have been made towards peer-review (by other architects) in a select set of companies, and this is also supported by an assessment drive by the SEI – these activities are discussed later. Contributing to the review-based improvement of the architecting process is the focus-area of this thesis.

In parallel, some emerging commercial development methodologies are emphasising architecture as a basis for development. This will at least raise the profile of architecture and in the future lead to tools, with hopefully accompanying investment into determining just what these tools have to support.

3.6.3.1 System qualities – neglected by most methods

As indicated by Bass, *et al*. [1997], there are two broad categories of qualities against which a system family (member) can be judged at the architectural level, those:

- observable via *execution* (or at run-time [Bennett, 1997]) – how well the system satisfies behavioural requirements in terms of correctness, performance, stability. Such issues are of importance for the normal *operation* of the system
- *not* observable via execution (or at build-time [Bennett, 1997]) – how well the system supports modifications, testing, reuse of development-resources, cost constraints. These issues can be regarded as more *strategic* in nature as they deal with business and life-cycle aspects of the system.

While the successful operation of the system is very important for long-term business success, and almost all aspects of the system will have some operational effects; the following table categorises some of the most obvious system qualities according to their *predominantly* operational or strategic nature. Definitions for these qualities are available in Bass, *et al*. [1997], and more details on the qualities selected for this research: interoperability and extensibility are provided in Chapter 4.

Table 3.1: strategic and operational system qualities.

Quality	Strategic	Operational
Functionality		✔
Dependability		✔
Performance		✔
Security		✔
Availability		✔
Interoperability	✔	✔
Extensibility	✔	
Modifiability	✔	
Portability	✔	
Reusability	✔	

In conventional software engineering, requirements and design contributions have focused almost exclusively on (user-oriented) functional requirements (see [Hammer, 1996] and [Bennet, 1997] pp12). Any attention paid to system qualities is largely restricted to those of the operational nature in Table 3.1 above, such as performance and availability (see e.g. [Kazman, *et al.*, 1998]) which are important for much of the embedded-software industry. As illustrated in Figure 3.7 below – attention to the more strategic system-qualities important for family-based development has been ignored by the majority of popular, commercial development methods.

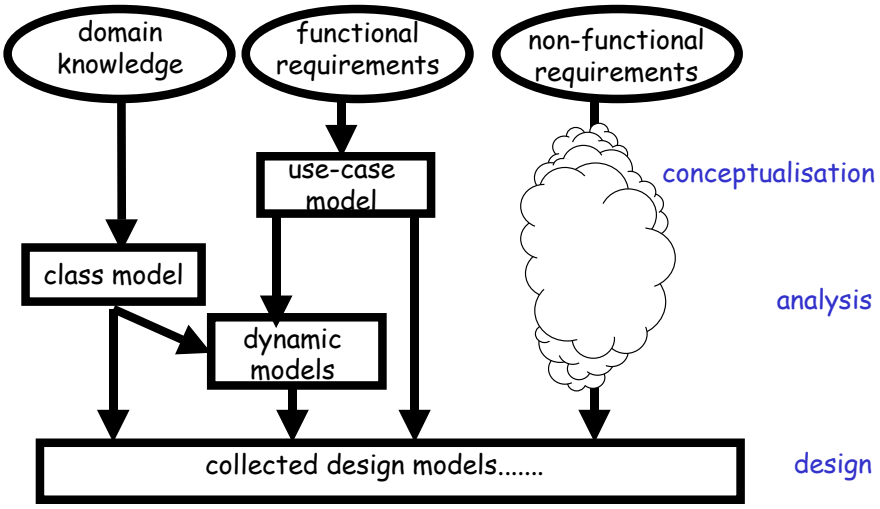


Figure 3.7: non-functional aspects (system qualities) an open area in modern development methods (adapted from [Warmer and Kleppe, 1996] pp.13)

As a result, the majority of information-system architecting occurring within conventional software development has been subject to the single-system approach with its focus on satisfying operational needs using organisation and technology resources (see Figure 3.8).

A method to assess an architecture with respect to some system qualities important for family-development is the contribution this thesis aims to make to software engineering practice. In doing so it will address some longer-term, strategic concerns; complementing the operational focus of conventional architecting.

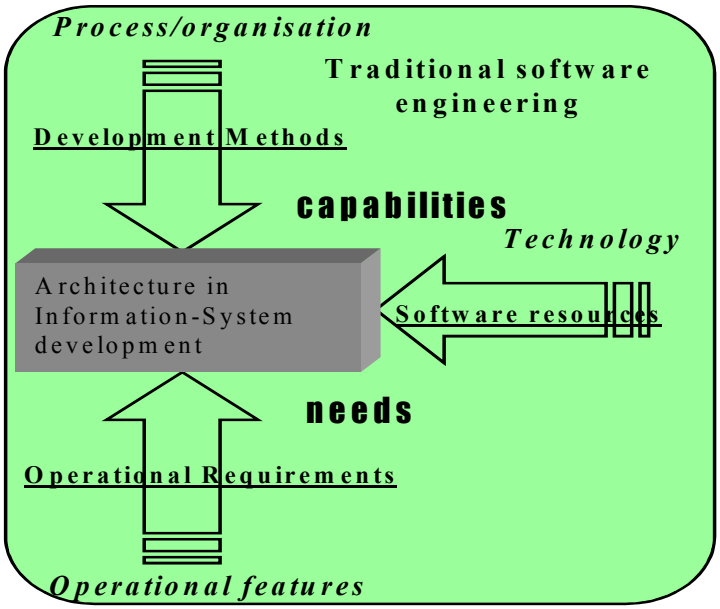


Figure 3.8: Driving forces on conventional information-system architecture

A requirements-rooted perspective is taken to the problem of product-family architecting. The stakeholder concept emphasised by Jacobson, *et al.*, [1992] is central to the treatment of architecture assessment to be pursued in this research. Much effort is needed to clearly involve stakeholders in the process and to help them to express their requirements as drivers for architecture development and evaluation. This concentration on the requirements associated with product-family system qualities addresses the gap in most of the conventional software engineering and architecting processes illustrated in Figure 3.7, and Figure 3.8 earlier.

The research aims not only to build on the current "popularity" of requirements and architecture – but especially to leverage the proven and practical implementations of these tenets from conventional tools and techniques. Further problem analysis and guidelines using these ideas are contained in Chapter 4.

3.6.4 Challenges

This section recalls a selection of those characteristics of the architectural artefact that pose challenges to product-family development. The subsequent section will identify those

research actions addressed in this thesis that are intended to ameliorate these difficulties. The challenges are:

- ***System qualities*** – The architecture can allow or preclude many important quality aspects of the system [Clements, *et al*., 1995]: modifiability, reusability, performance, interoperability, and is therefore of major importance in enabling effective product-family development. However, explicit support in dealing with these qualities is weak [Hammer, 1996] in the traditional development methods underlying the architecting process (see 3.6.3.1).
- ***Stakeholder communication*** – Product-family development is a communication-intensive activity between multiple disciplines, who must consider demands from current, past, and future conditions. A design artefact as crucial to the commercial success of the family as the architecture should support this communication in a way that is both accessible and visible.
- ***Architecture is unknown*** – Architecture is a critical ingredient for product families yet due to the:
  - complex trade-offs that must be made in architecting;
  - the highly personalised role of the architect;
  - the inapplicability of a traditional scientific method;

  architecting has developed an air of mystery; and the architecting process is largely left to the architect – critical examination prior to implementation is rare. This presents a large risk for product-family management.

3.6.5 Potential solutions

While recognising that developing formal rules to arrive at architectures is impossible; what is feasible, and important, is that the architecture should be made as visible as possible, so that it can be communicated to, and assessed by, the various development stakeholders to ensure that it conforms to the family strategy. In particular, the case for capturing the rationale behind architectural decisions, from a requirements perspective, has been made by many sources [Gacek, *et al.*, 1995], [Bennett, 1997], [Bass, *et al.*, 1998]. This will provide a reference from which to debate and judge architecture and to manage its creation and evolution in the context of the requirements driving the family.

The research described herein recognises the increasing importance of, and interest in, architecture in the industrial and research communities. The motivation is to reflect the importance of the architecture to product-family development by raising its profile within the development organisation and making it a "living" artefact that is explicitly used in the operational activities of the development team.

Assessment is a recent trend in architecting (see [Kazman*, et al.*, 1996]) and can be regarded as a mechanism to realise the aim of increasing architectural visibility. This activity offers

promising results in getting the architecture debatable among the wider group of architecture (and family) stakeholders.

Developing a method for assessing family architectures against family-relevant system qualities is the aim of this thesis. It intends to deliver on the promise of increasing architecture clarity, and to do so by:

- concentrating on issues of particular importance to family-development;
- adopting assessment as part of an iterative approach to architecture development;
- providing techniques to enable stakeholders (the customers of the architecture) to be fully involved in the assessment process.

The issue of architecture assessment will be covered in the next section.

3.7 Architecture assessment

As stated, this research work aims to develop an assessment method enabling family stakeholders to assess information-system family architecture for conformance to system qualities important for their family. It is appropriate at this point to provide some background on the general application of assessment in information-system development and later proceeding to an analysis of how architecture assessment is currently practised. The section will close with a proposal as to how architecture assessment can be used in this research to improve architectural quality.

3.7.1 What?

The term "assessment" was first introduced into the software engineering literature by Crosby [1979] in relation to the determination of certain qualities in a process. Assessment is regarded in this thesis as the act of verifying to what extent an entity satisfies certain requirements (the assessment criteria) placed on it. This is a generic approach, reflecting the generic character of assessments and their treatment in literature from informal document reviews to rigorous certification of process [Humphrey, 1989]. Many management and quality practices are based on the 4-step: "Plan-Do-Check-Act" [Demming, 1982] principle – assessment is a tool to address the check-step.

The ISO-9126 series of quality standards also take a requirements perspective and **define** evaluation[13] as [ISO-2]:
the systematic examination of the extent to which a product or service fulfils requirements.
Instantiating the generic "product or service" with "architecture" provides the definition of architecture assessment used in this research.

[13] The quality literature does not consistently distinguish between evaluation and assessment and the terms are interchangeable – in this thesis "assessment" is used as it is the more established term in the general software engineering, and the software architecture literature.

A more detailed impression of assessments in the software industry is provided when considering the rationale for, and practice of, assessments in the following sections.

3.7.2 Why?

Structured assessments are a standard part of the software engineering development process, and are an empirically proven means to improve the quality of a software artefact (see [Fagan, 1986] talking about inspections). Assessments can be broadly classified as:

1. **improvement-oriented** – typically these are done internally in the development team using intermediate work products, and the aim is to provide feedback so that improvements can be incorporated. The assessments can be formal against specified criteria (e.g. tests), or can be less-structured with criteria established during the assessment (e.g. Fagan inspections, code walkthroughs). These assessments may work with incomplete information in an incremental development environment (see e.g. [Fowler and Scott, 1997], Chapter 2) and therefore may be repeated regularly. These assessments are of the "what can we do better?" type.

2. **certification**[14]**-oriented** – typically these assessments are conducted by third-parties (to the development team), often by professional institutes and the aim is the achievement of a formal certification indicating compliance to the pre-defined assessment criteria. Criteria and assessment process is typically well-structured, fixed and formal. Examples are standards-conformance for products, and CMM-assessments for process [Humphrey, 1989]. These assessments are typically applied infrequently in a development cycle and usually when there is a strong chance of success, therefore information (and sometimes product development itself) is complete. These assessments are of the "what have you done wrong?" type.

The assessment method which is the subject of this research falls into the first "improvement-oriented" category.

The early assessment of architecture against the requirements is advocated in this thesis as a means of ensuring the suitability of the architecture. The timely determination of architecture quality is important because:

- the pivotal role that architecture plays on the life cycle of any software system means that mistakes in this phase are far-reaching: typically 50%-70% of life-cycle costs are determined by architectural choices (studies quoted in [Bengtsson, *et al*., 2000]);
- it is accepted [Jacobson, *et al*., 1992] that the cost of repairing errors increases an order-of-magnitude for every life cycle phase (requirements, design, implementation, testing)

[14] Certification has, of course, the implicit aim of improving quality; but the most obvious intention of a certification-based assessment is to get certified.

that the error remains. The requirements-design transition is an ideal opportunity to avoid propagating errors throughout the larger development organisation.

The scale-effects of architecture failures are magnified in a product-family development environment, making the assurance of architecture quality especially important here.

In addition, a formal assessment can serve as a vehicle to encourage both stakeholders and architects to be explicit in their requirements and architectural descriptions, and will facilitate the early discussion of important family issues – which is inherently beneficial to the team's effort.

It is important to realise that the requirements-driven architecture assessment method as proposed here is one of a collection of quality-improving efforts throughout the product (family) development life cycle. There are other more well-established software-engineering concepts such as testing and prototyping used as part of a battery of techniques used to align and assess software development entities with respect to preceding entities. This is depicted in Figure 3.9 below which provides a context (based on the logical V-model of development[15]) for architecture assessment in relation to the more established quality methods. The important points are:

- the architecture assessment is the *earliest* check in the development cycle;
- it does not need *investment* in detailed design or coding.

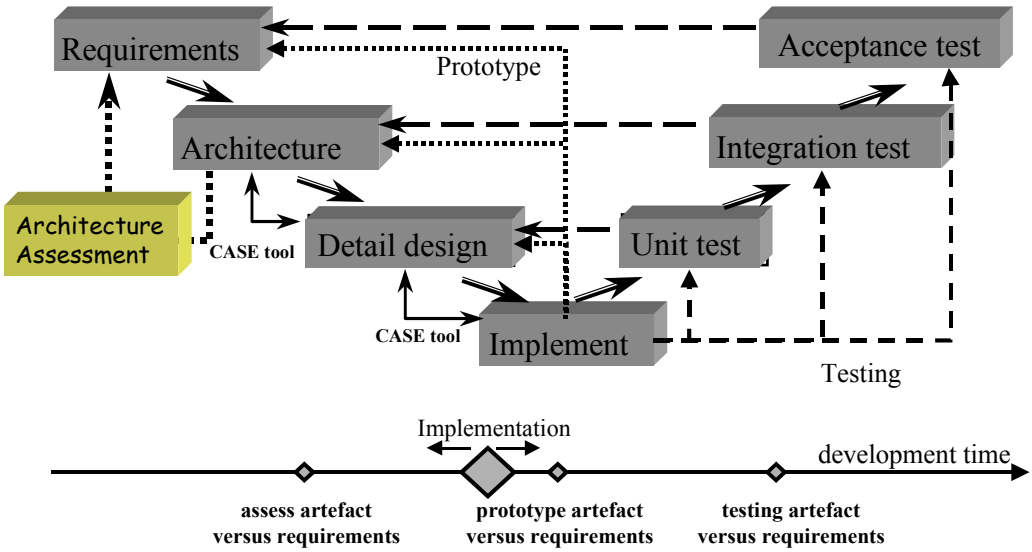


Figure 3.9: Architecture assessment in context of (logical) development process

[15] Even if incremental development is pursued – the logical-correctness of the v-model is still preserved. Coding cannot (should not!) be completed before design, integration cannot be completed before unit-testing. Incremental development provides a means to begin (and hence end) each major activity earlier by focussing on partial functionality in each iteration – but the principle remains: assessment of requirements against architecture can still occur before assessment against integrated code.

In relation to product-family development; the architecture is the *single* entity that captures the full scope of the family – subsequent development entities will (especially those realised in code) incorporate variance through multiple, separate versions, and may simply not be available until the distant future. Assessment of the architecture is thus crucial to the family as it is the only family-wide entity that can (and should) be tested early in the development process.

Of course there are no "free lunches" here; the speed and cost advantage must be weighted against the fact that a purely requirements-driven architecture assessment will *not* guarantee a completely successful product. Errors may still be introduced; the architecture may prove to be misaligned with the coding platform or the organisation; other implementation issues may arise which invalidate the system. The point is that existing code-based assessments are still necessary. This work addresses another development aspect (architecture) which is important and currently not adequately supported.

So although architecture quality is typically not an explicit subject in the mainstream software quality literature or practice, there are positive developments. The architecture-oriented literature that has emerged in the last 5 years, while recognising that formal system requirements are not the only input to architecture, have continually emphasised the importance that requirements play in ensuring architecture quality. Rechtin and Maier [1997] in their comprehensive treatment of architecture have also stressed this point. The following section will discuss the progress in current practice.

3.7.3 How?

A striking feature of the existing methods for product assessment is that they are largely *code-driven*, i.e. they are based on code being exercised and the resultant behaviour compared to that specified or expected in earlier development entities. This has the great advantage that you are assessing (close-to) the final product. The production of suitable code is what the software-development process is primarily about, and no-one would realistically conceive of releasing or accepting untested code. The disadvantage of exclusively code-based assessment is that one has to make the investment in *time* and *money* of developing the code before one can assess the alignment of the code with the preceding entity. Further misalignments of code and e.g. requirements may have been injected into the process at the architecture, detail-design, or coding stages, so extensive detective and re-design work may have to be done to discover and correct the faults.

The requirements-driven architecture assessment method advocated in this research aims to provide a cheaper, faster assessment loop so that stakeholders are satisfied that at the end of the architecting phase all important requirements have been addressed, and no misalignments between requirements and architecture exist. For architecture assessment, the *Quality = conformance to requirements* [Crosby, 1979] approach adopted by this research is the only

feasible approach for early-phase architecture assessment. This is because architectures are mainly developed in the early, pre-prototype, system-development phase; and where prototypes are available they are usually used for a restricted, performance-related aspect. Requirements are the only appropriate, and only existing artefact against which to evaluate the architecture.

In this sense the assessment method provided by this research addresses a gap between:

- high-level, and product-independent, process assessments;
- low-level (code), (single-)product-specific, late-life-cycle product assessments;

by concentrating on mid-level (design), product-general (family), early-life-cycle product family assessments.

An analysis of current architecture assessment methods is included in Chapter 4 when the topic and the solution principles underlying the research are discussed in detail. In the remainder of this section, some challenges and their potential solutions are indicated, again this potential will be developed further in the following chapter.

3.7.4 Challenges

Important challenges to requirements-based architecture assessment include:

- ***Document requirements*** – The basic idea running through Bennett's [1997] book, which deals with the entire software development cycle, is that evaluation of design documentation against documented external goals (requirements) is critical for software development. The challenge here is for stakeholders, analysts, and architects to proactively document (or seek documentation for) all the external goals, constraints which influence the architecture.
- ***Stakeholder involvement*** – Those stakeholders responsible for the family requirements driving the architecture, particular for system qualities, have been ignored by conventional development and assessment methods. Both stakeholders and architects need practical aids in establishing the criteria for assessment and preparing the architecture to be assessed.

3.7.5 Potential solutions

There are some potential solutions to meet the challenges presented above in a collection of architecture assessment methods undergoing industrial application. In particular a requirements-based assessment method involving stakeholders is being promoted and developed by the SEI, and provides a starting point for the problem area addressed in this research. These methods and their application to the domain of information-system families is the subject of the next chapter.

3.8 Elaborated problem statement

The research problem has been presented in Chapter 2, and it is elaborated here in the light of the detailed review of the problem domain.

Based on the survey of the problem domain above it clear that several challenges remain unresolved in developing information-system families. In particular it has been shown that the non-functional aspects of systems – the system qualities – are of prime importance to the ability of a family to address markets and evolve with them.

Architecture has been recognised as a concept which can help to manage the complexity of understanding the family, and which operationalises the commercial strategy so that the technical basis of the family enables the desired requirements within the operating constraints of the firm. In particular, because of its system-level approach, architecture is determinant in providing the system qualities necessary for family competition.

Contemporary software development practice and methods have not devoted sufficient attention either to:

- addressing the system qualities;
- supporting architectural issues.

Architecture itself is similarly a complex and immature domain – with initial steps into codifying professional knowledge only recently initiated. Architecture is, and will remain, a largely people-centric discipline.

Recognising the above, the work in this thesis is:

1. focused on addressing the gap in current software development of product families with respect to system qualities and the role of architecture therein;
2. based on realising that architecture, and information-system family development are complex, human-centred processes.

There is no pretence that prescriptive models can be generated. The aim, therefore, is to enable the various family stakeholders to constructively interact with the architecture as a means of reducing the risk that serious family-architecture mis-matches occur.

Summarising, the problem to be addressed in this thesis is:

> How can the stakeholders assess the information-system family architecture with respect to important system qualities (i.e. interoperability, extensibility)? – so determining that the architecture is addressing their requirements, and hence supporting the product-family based business?

The research deliverable is an **assessment method** (guidelines, metrics, recommendations, and process) addressing the above problem. The remaining chapters provide an analysis, design and implementation respectively of the method.

3.9 Summary

In this chapter the problem domain and its main characteristics have been described based on observations and analysis of literature and practice. Definitions for information-system families and architecture have been provided, and an elaborated problem statement has been formulated.

It has been shown that the major challenges in product-family development are supporting the necessary multi-stakeholder discourse to ensure co-ordination, particularly with respect to the system qualities, which are very important for family life-cycle management but have been ignored by mainstream development methods. The importance of architecture as means to co-ordinate stakeholder communication and family evolution has been discussed. Architecture is a much-talked-about term, and the discussion is still ongoing, however, a working definition of information-system family architecture has been provided.

Architecture practice is currently very much an architect-driven process, and formal review, understanding of the architecture in the organisation is typically very limited. Considering the importance of architecture this is a high-risk practice. The value of an architecture assessment method that will expose the architecture to stakeholder-review with respect to the system qualities is seen as a valuable contribution to practice – and is the problem that will be addressed in the remainder of the thesis.

The following definitions have been provided in this chapter, and will be used throughout the remainder of this work:

*a **product family** is product concept that is designed for a market but caters for the individual wishes of customers by introducing variety within a defined product architecture*…[Erens, 1996].

Information systems: "*Computer-based systems intended to provide recording and supporting services for organisational operation and management*". [Verrijn-Stuart, 1989]

*an **information-system family** is a set of information systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment or customer; built from a common set of core assets*. (adapted from [Bass, *et al*., 1998])

***Information-system family architecture** is a collection of models, each reflecting a high-level structure of the information-system family's generic components, showing the externally visible (to other components) properties of these components and their relationships. This*

family architecture is used to derive particular system architectures for individual family members.

Architecture assessment is *the systematic examination of the extent to which an architecture fulfils the requirements placed on its associated product or service*. (adapted from [ISO-2])

Chapter 4
Analysis – architecture assessment

> "*Systems will be modular, easy to extend, and robust in the face of change only when their designers explicitly evaluate alternative designs for modularity, ease of extension and robustness against explicit requirements for growth and change*." – [Bennett, 1997], pp237

4.1 Purpose

The purpose of this chapter is to further analyse the concept of architecture assessment for information-system families. This analysis concentrates on:

- obtaining a comprehensive overview of the current approaches and issues;
- identifying opportunities for improvement, based on the review in Chapter 3;
- scoping the problem area to be addressed by this research work;
- establishing important requirements for the method to be developed.

In doing so, the first of the research questions from Chapter 2 will also be addressed in more detail.

4.2 Structure

The chapter is organised as follows: Initially an analysis framework is introduced that provides a conceptual context for positioning the various issues (challenges, techniques, resources, and roles) associated with architecting family-based information systems. Subsequently, the practical application of architecture assessment in software development is presented. The SEI-backed SAAM method will receive most attention, but other methods will also be briefly discussed. The section concludes with an overview of opportunities to improve on the current state of practice, particularly with respect to addressing the previously identified challenges (see Chapter 3) to product-family architectures.

The scope and context of the assessment method to be developed in the research, is addressed in the second part of the chapter. The focus on the strategic aspect of information-system family architecture and in particular the system qualities of interoperability and extensibility is explained. The first research question from Chapter 2, accommodating with the primary information-system family stakeholders, is addressed. An overview of stakeholders and their concerns from literature is provided and followed by pragmatic guidelines to identify the core stakeholders to be involved in assessments. Hereafter a selection of high-level requirements, assumptions and constraints within which the method will operate are presented; these will drive the method design described in Chapter 5. The chapter concludes with a summary.

4.3 Analysis framework

4.3.1 Introduction

This research is motivated by two facts that are widely accepted (recalling the previous chapter) in the software development research, and industrial communities:

1. Architecture is a key determinant of successful product-family development;
2. The explicit recognition and support of stakeholder concerns is regarded as an integral part of modern software-development methods.

It was also shown in the previous chapter that support for family stakeholders in:

- expressing important system qualities as input to the architecting process;
- translating these qualities to architectural properties and so enabling interaction with the architecture to verify commercial and technical alignment

was deficient. An architecture assessment method was proposed as a solution principle towards addressing the above deficiencies in current practice.

In order to organise the various strands of knowledge associated with the research area, a research framework has been developed. This depicts the main perspectives from which to regard information-system architecture and illustrates the context for the research focus dealt with in this thesis.

4.3.2 Research framework

A framework has been developed (see Figure 4.1) that separates the main viewpoints from which the research object (information-system family architecture) can be regarded. The aim of this separation is to communicate both the *scope* and *context* of the research and to facilitate problem analysis through separation of concerns. This identification of viewpoints, based on the main forces shaping architecture, is also useful in identifying knowledge sources that must be investigated to adequately address the research area. The framework is organised around the themes of needs (left and bottom parts of the figure), and capabilities (top and right parts of the figure). As described in Chapter 3, and shown in Figure 4.1 below, software architecture is conventionally regarded as describing how the needs of the various *operational* system stakeholders (bottom arrow) are satisfied using software technology resources (right arrow). In addition, the development view of architecture provides a set of structures for organising system development and maintenance in the context of a product development process using the techniques, practices and people of the organisation (top arrow).

This traditional depiction of the main forces influencing architecture has been extended to account for those *strategic* (long-term accommodation of change, and evolution of assets) needs of the market and the firm (left arrow) manifest when information-system families are being developed. These needs are expressed as system qualities, and are the concern of those strategic stakeholders interested in the business aspects of the family. The figure also

indicates that family development will impact on the other forces; e.g. the strategic aspects of technology will become more important, and the development organisation may undergo significant change. These will not be dealt with further however.

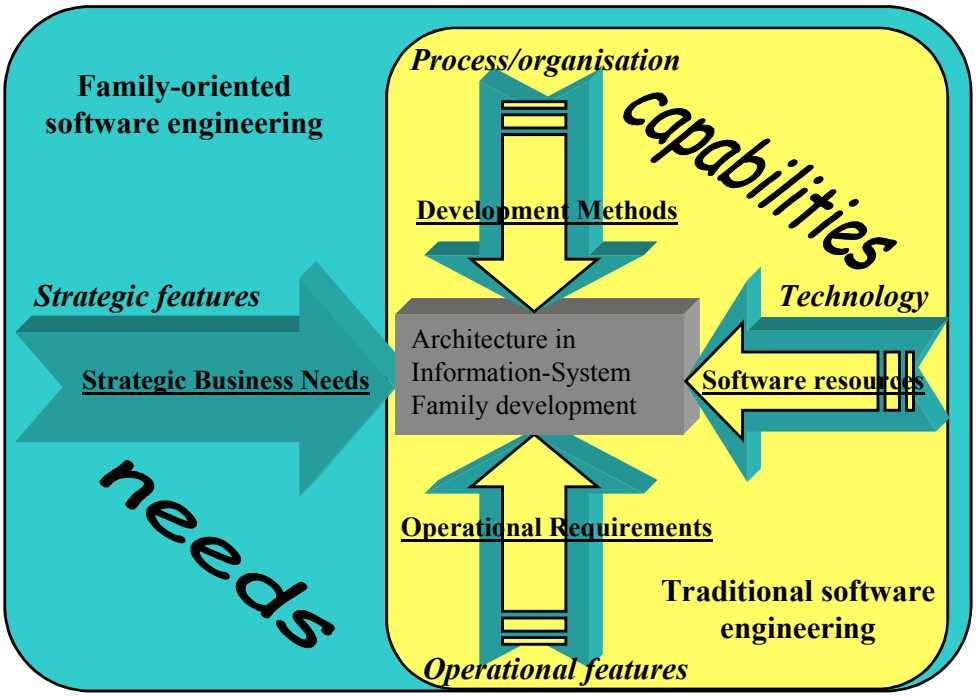


Figure 4.1: Research Framework for information-system family architecture

The framework introduced here is comprehensive in breadth. It includes needs and capabilities both internal (operational requirements and development method) and external (industry needs and state-of-the-art technology resources) to the company. Central to both internal and external aspects of family development is the architecting process and the architecture artefacts that support it. The aspects are further illustrated by listing sample issues that need to be addressed by information-system family architecture, including:

- **Technology** – communication standards (which are key solution providers to the demands for openness and interoperability) are regarded as software resources;
- **Process/organisation** – development methods must support architectural descriptions and the overall development process which encompasses the architecture. The skills of the organisation must be mirrored in the technology choices prescribed by the architecture.
- **Operational features** – the requirements of the users for functionality and usability must be addressed by the architecture. This aspect is dominant in system conception and development and receives the most attention from development methods and participants.
- **Strategic features** – this relates to the long-term issues associated with the continued success of the system family. The stakeholders have a strong business-orientation, and are concerned with the ongoing evolution of the family members both in the customer's, and producer's business context. Typical concerns are extensibility, portability, reuse, and interoperability.

The graphical framework depicted in Figure 4.1 is related to Gacek's [1995] broad definition of software-system architecture presented in Chapter 3 (and repeated here for readability) – a ***software-system architecture*** *comprises:*

- *a collection of software and system components, connections and constraints;*
- *a collection of system stakeholders' need statements;*
- *a rationale that demonstrates that the components, connections, and constraints define a system, that if implemented, would satisfy the collection of system stakeholders' need statements.*

The architecture elements indicated in the first point in the above definition are associated with the technology aspect (right arrow); their representation and implementation is part of the development method (top arrow) aspect. The operational requirements aspect (bottom arrow) and the strategic business needs aspects (left arrow) with their focus on the various system stakeholders cover the second part of the definition.

The rationale – illustrating how the architecture addresses the requirements – is the focus of this research activity. Rationale involves explaining and assuring stakeholders as to the suitability of the architecture in meeting their requirements. An assessment method is a technique directly applicable to providing architectural rationale and hence satisfying the definition above. Assessment is a key concept in architecture development.

The separation of requirements into those operational requirements associated with user and single-system development (bottom arrow) and those more strategic requirements associated with the market and firm (left arrow) is very important for this research. The latter requirements represent those system qualities earlier identified as important for product-family development i.e. long-term, business- (as distinct from function-) related and spanning multiple products (and hence development projects). This research recognises that strategic, family-related system qualities have not received adequate support from conventional methods.

The added value of the research is in redressing this imbalance by providing a practical method to express and assess architectural rationale thereby ensuring that quality (in the sense of conformance to requirements) information-system families are developed.
The main focus of this research work is assessing how well the information-system family architecture addresses the system-quality requirements of the strategic stakeholders; and therefore *the proposed assessment method will concentrate on this aspect.*

The value of the framework is in providing a conceptual structure within which to organise all the various issues associated with architecting software-system families. It also provides a context so that the research contribution (architecture assessment method) can be positioned for what it is (and what it is not) in the "big picture" of information-system family

architecture. In this regard, it is clear that the research effort concentrates on the left arrow of Figure 4.1, further refinement of the research focus will be provided later in this chapter. The following section provides an overview of architecture assessment practices in industry.

4.4 Architecture assessment methods in practice

While formal assessment of software artefacts have been reported since 1972 [Fagan, 1985], it is only in recent times that architecture has been publicly subjected to critical analysis [Kazman, *et al.*, 1996]. The SEI has studied the state of practice of architecture evaluation [Abowd, *et al.*, 1997] through annual workshops with invited practitioners since 1996. Among their findings they categorised assessment techniques, see Table 4.1, into *question*-based (first three rows) and *measurement*-based (last two rows).

Table 4.1: Properties of assessment approaches (from [Abowd, *et al.* 1997])

Basis	Method	Generality	Detail (arch.)	Phase (arch.)	Object
Question	Questionnaire	General	Coarse	early	artefact / process
Question	Checklist	domain-specific	Varies	middle	artefact / process
Question	use-cases	system-specific	Medium	middle	artefact
Measurement	Metrics	general / domain-specific	Fine	middle	artefact
Measurement	Prototype, simulation	domain-specific	Varies	early	artefact

The table is reasonably self-explanatory, however it is worth clarifying:

- the difference between **questionnaires** (very general, reasonably open questions) and **checklists** (more detailed domain-focused questions) is that the latter are more domain-dependent than the former. Both are associated with a mature architecture assessment process, where practice is standardised sufficiently to be captured in accepted questions and checklists.
- **Use-cases** are typically system specific and represent context-based descriptions of "ilities"; they are also typical of less-mature assessment practice, where knowledge is insufficiently stable or complete to be formalised into questions or checklists.
- **Metrics** are quantitative and are typically associated with fine-grained architectural descriptions (i.e. more architectural details have been established, e.g. inter-module call structures, hardware allocation) which are determined later in the architecture definition.
- **Prototypes** can provide concrete quantitative results and typically address a single "ility"; alternatively they are used to investigate early architectural principles.

The message from Abowd, *et al*., [1997] is that there are multiple approaches each with different emphasis on some aspect of the architectural development process. Appropriate techniques for a particular assessment depend on the development process, maturity of evaluation within the organisation, and the qualities being examined. In general they advise

that assessment be a planned activity in the development cycle as an aid to institutionalising architecture-based quality.

Regarding architecture assessment best practice, the same authors have provided some additional points of information:

- **Questionnaires** (general) and **checklists** (more specific) usually exist before architecture is developed (analogous to building codes), and follow from extensive experience in the domain. They are indicative of mature assessment processes.
- **Use-cases** are used to provide context to qualities and are geared towards specific systems (or system families). Use-cases are developed during the assessment, and are typical of an immature assessment process. Experience with use-cases across a family of related systems might be used as seed-material for deriving more generic checklists, and questionnaires.
- **Metrics** are a measurement-based assessment technique – they should be used with caution because metric definitions have underlying assumptions (not always explicit) which must be perceived as valid for the particular assessment artefact. Furthermore, the predictive power of some quantitative quality models (e.g. reliability, see [Brombacher, 1992]) is questionable despite their formality.
- **Prototypes** are the most visible means of assessing architecture – but are most suited to performance-related aspects; typically prototypes are defined to address a particular area of concern and cannot provide much information outside that scope.

The assessment method to be developed in this research belongs to the use-case category. This is appropriate because the domain of information-system family architecture is well suited to the application of use-cases:

- ❑ the problem domain is immature, both in terms of building product families and in evaluating architectures;
- ❑ extensive stakeholder interaction is needed during family (architecture) definition, and stakeholder dialogue is naturally facilitated through use-cases;
- ❑ the use-case-based approach, with its concentration on systems (instead of generic domains) and its applicability to coarse-grained architectural detail is suited to the "collection of existing-systems" (a suite, see Chapter 3) approach common to practical family definition.

The remainder of this section will provide more detail on the most widely reported use-case based assessment method to date – SAAM. This method is also officially backed by the SEI as an integral part of their developing Software Architecture Initiative strategy. In addition to being non-proprietary, the method is undergoing active extension and adoption by third-party researchers. It is therefore likely that the method will become widespread and perhaps eventually standardised by the SEI. With this in mind, SAAM provides a solid basis on which to base the assessment of product-family architecture.

4.4.1 SAAM

The SAAM (Software Architecture Analysis Method) is a stakeholder-centred, use-case (scenario) [Jacobson, *et al*., 1992] based assessment method (see [Kazman, *et al*., 1996]) intended to analyse architectures with respect to various non-functional system qualities. SAAM has been described and reported on extensively by the SEI ([Kazman, *et al.*, 1996], [Clements, *et al.*, 1995], [Abowd, *et al.*, 1997]); the main aspects of the method are reviewed here. The review is organised according to the following key aspects of the method:

- Goal;
- Assessment criteria;
- Representations;
- Process, people and roles;
- Overview.

A Note on Terminology: Use-cases are generalised scenarios (specific instances of goal-driven, actor-system interactions). "Scenario" is the term adopted by Kazman, *et al*., [1996], but "use-case" is the accepted term in the general body of software engineering literature; they are regarded as equivalent in this work. SAAM literature refers to "scenario" and, in the interests of compatibility with that literature, this term will be used when describing SAAM details. Elsewhere the term "use-case" will be used for compatibility with the general software engineering literature.

- **Goal** – SAAM concentrates on a selection of system-qualities (similarly to this research). Its primary use to date has been in the early-phase of architecture development, or where more-mature architectures are being validated in procurement or upgrade contexts. It has focused on enabling a process by which participants can draw conclusions about the architecture. A distant aim is to measure architecture against a quantitative scale.

- **Assessment criteria** – The assessment criteria are used to determine how well the architecture supports the various stakeholder requirements. Requirements are related to system qualities that are regarded as important for the system. In order to operationalise vague system qualities such as "modifiability", SAAM encourages reviewers to propose specific, representative, cases (scenarios) illustrating the desired behaviour that the system must support. This addresses the real requirements behind the quality. There is little value in the question "Is this system modifiable?" because the answer is context dependent. The correct approach is to determine what you want to modify and then ask, "How will the system accommodate the following change[1]?" (see [Kazman, *et al.,* 1996] for a more detailed discussion on this topic).

[1] This focus on identifying the desired change, has led to the "use-cases" being termed "change-cases" for the method developed in this research, see 4.6.2.1.

There is no quantifiable score on how well a scenario is satisfied, as this is context-sensitive. SAAM does, however, allow some metrics indicating the degree to which components are related to multiple scenarios, generally an indication of poor isolation of functionality [Kazman, *et al.*, 1996].

- **Representations** – SAAM adopts a scenario approach towards representing and eliciting stakeholder requirements in relation to the system qualities. In practice for SAAM it is usually sufficient to clearly identify the scenario by name, a structured specification of the desired system change is not provided.

  In principle SAAM leaves the question of what architecture representations (it is stressed that multiple views may be needed) to use and the level-of-detail needed, open to the method participants however, they advocate the 4+1 views proposed by Kruchten [1995]. In particular, they have found that the *logical* and *development* views proposed by Kruchten are appropriate for a wide range of build-time requirements [Kazman *et al*., 2000].

- **Process, people and roles** – The SAAM process is based on examining the architecture to see to what extent it supports each scenario; where architecture-modifications are required – these are associated with the affected component, and an estimate of the consequences of the modifications is made. The results are combined with those for the other scenario and the team decides on whether to implement the modifications or to postpone, or change, the requirements. The following steps are used in the method (see Figure 4.2):

1. Describe the candidate architecture – (done iteratively with step 2);
2. Develop scenarios – appropriate for all stakeholders;
3. Evaluate scenarios – determine what architectural changes are needed to support the scenarios and estimate consequences of implementation;
4. Reveal scenario interaction – identify changes to components from multiple scenarios; this identifies critical components and potentially signals low modularity;
5. Overall evaluation – the scenarios and their interactions are weighted in terms of relative importance and this rank is used to determine what overall actions shall be taken to address the requirements.

SAAM is generally guided by an external facilitator: the stakeholders are polled for scenarios, and during evaluation the architect is asked to illustrate (using architectural views, documents, verbal presentation) how the scenario can be accommodated. The team decides on the course of action to be pursued after the review.

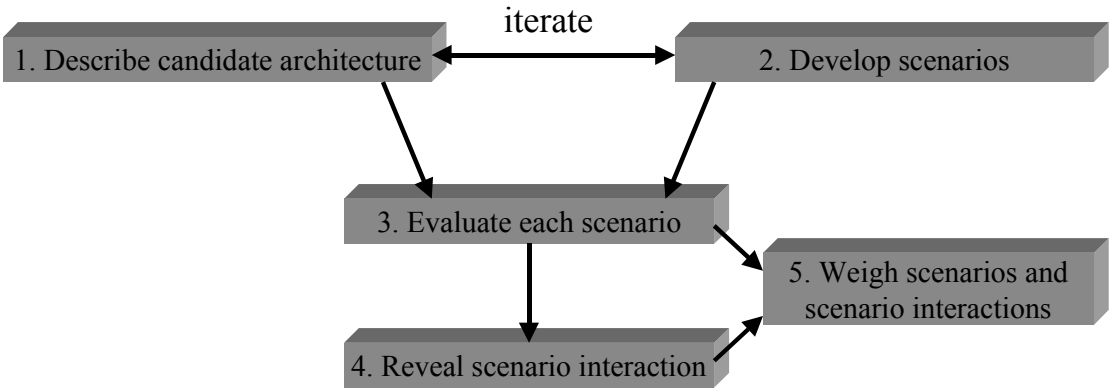


Figure 4.2: The five steps of SAAM and their dependencies (from [Kazman, *et al*., 1996])

- **Overview** – SAAM is a straightforward, yet powerful initiative towards institutionalising architecture assessment. It has been applied in a wide variety of projects and industries and is actively being supported and extended by the SEI and partners. Its main benefits are in illuminating architectural properties early in the life cycle, through which the development team detects and resolves potential problems as early as possible. Considering the current state of architecture maturity SAAM benefits are largely confined to people and process [Kazman, *et al.*, 1996] and include:
  - deepened understanding of the system;
  - enhanced communication among developers, developers-managers-customers;
  - ability to make high-level comparisons of competing designs across sets of changes and to document those comparisons and changes.

SAAM has also motivated a more-advanced method; Architecture Trade-off Analysis Method (ATAM) to address the fact that architectures must deal with tradeoffs between competing qualities (i.e. multiple qualities must be analysed and the results aggregated, see [Kazman, *et al.,* 1998]).

4.4.2 SAAM – a critique

The following aspects of SAAM provide opportunities for improvement:

1. SAAM is not focused on families of systems (although does not exclude them, see [Kazman, *et al*., 2000]), and most reported experience is on single-system architectures, and addresses the roles of operational-stakeholders from the user or development communities. SAAM has also focused on a subset of the important system qualities, most notably security, adaptability, portability. Experience of **family-oriented assessment**, particularly in relation to other **family-relevant qualities**, therefore represents a useful contribution to the assessment experience base.
2. SAAM needs scenarios to be generated (step 2 of the method) – but does not provide any detailed guidelines as to *how* they might be *identified*, *generated*, *represented*, or *prioritised* – this is particularly important for strategic system qualities as there is no broad experience base from which the stakeholder can draw upon. Providing **guidelines** and **techniques** in this area is, thus, a *necessary* contribution towards increasing the

adoption, and usability, of the architecture assessment method in industry.

3. SAAM was originally intended to compare candidate architectures, but is also very applicable towards establishing increased stakeholder understanding of a single architecture. It is also being used early in the requirements analysis phase of development to assess the impact of requirements, therefore providing a valuable tool in requirements negotiation and perhaps in family definition. Extending this through a **more intensive stakeholder involvement in the method** is particularly appealing for product-family architectures, where the negotiations are strategic and the opportunities for communication more important.
4. The majority of reported SAAM experience is in the one-off, externally driven, certification-type (see Chapter 3) assessments typical of acquisition, or supplier-selection, oriented reviews where the assessment sponsor is the customer (e.g. Chapter 11 and Chapter 9 from Bass, *et al*., [1998]). Most software families are commercially-driven undertakings, made by a (single) producer for multiple customers (see Chapter 1). Supporting the producer in evolving his architecture over time to address his market by means of **incremental, improvement-oriented *self*-assessments** is more appropriate to product-family development, as distinct from product-family procurement.

This research will address the above issues and thereby build-on and usefully extend and experiment with an emerging *de facto* assessment standard[2].

4.4.3 Other industry methods

The current state of practice with respect to non-SAAM based methods is largely confined to:

- in-house developed methods by large software-oriented firms (e.g. the SARB, Software Architecture Review Board of Bell-Labs);
- architecture product or process assessments developed and sold by consulting companies (e.g. the Architecture Improvement Service developed by Origin [Postema, 1998]).
- Emerging assessment approaches from research institutes (e.g. the ALMA approach of Lassing *et al*., [1999] and the profile-based assessments of Bosch [2000] in his QASAR architecture design method.

All types are strongly oriented towards self-assessment (e.g. Origin) and/or peer review with other architects (e.g. SARB). Reported industrial experience of these assessment approaches is relatively limited, although SARB is being widely used within Bell-affiliated companies [Weiss & Lai 1999]. The remainder of this section will provide a (very) "rough-guide" to these methods mainly based on literature and second-hand "travellers-tales" from practitioners who have been involved.

[2] This may seem speculative at this time, when architecture assessment is so immature, but the facts that there is a dearth of non-proprietary alternatives, the SEI is using it as part of their software architecture strategy, and the increased adoption of the method by other research-schools (see [Briand, *et.al.*, 1998]) is strong evidence that SAAM-concepts will not disappear in the medium term.

Again system-quality assessment is generally the **goal** of these methods, although some are also extended to analysing how technology is used (i.e. the technology aspect of the framework presented in Figure 4.1) in the architecture. Assessment **criteria** are (as with SAAM) based on stakeholder-relevant requirements, but again the detail, and precise role of the stakeholders in establishing these requirements varies. The architects, for example, generate the requirements in the QASAR-assessments [Bosch, 2000].Tools to aid criteria definition are typically scarce, checklists occasionally appear, but in some cases are very generic (especially with consultancy-based methods which are usually pitched to operate in multiple industries) and are experienced to be of limited use because of the high level of domain focus of architectures. The ALMA-method [Lassing *et al*., 1999] uses a risk-oriented categorisation matrix to ensure use-case coverage, especially of complicated use-cases. Metrics are scarce and where used are typically confined to architecture (not requirements), an example are the effort estimates for maintenance described in [Bengtsson and Bosch, 1999].

Requirement **representations** are typically short sentences reflecting sample use-cases in SAAM-style, or even single words representing the desired quality. Simple "one-liners" stating the fact that e.g. modifiability should be supported (recall the earlier discussion in section 4.4.1 above on the usefulness of such an approach) are still common in many of the consultancy-based assessments. Architecture representations are also variable, reflecting the diversity common in industrial practice. Increasingly some of the well-accepted views mentioned in Chapter 3 e.g. [Kruchten, 1995], [Soni, *et al.*, 1995], are observed.

The assessment **process** is, in general, SAAM-like in the sense of: agree scope, get quality criteria, get architecture, assess, report. The roles of people in the assessment differs widely:

- in SARB the assessment team is largely composed of experienced architects from outside the development team (this "external-assessor"[3] line is also advocated by the SEI, [Abowd, *et al.*, 1997]);
- in the Origin method, the assessment is primarily from within the development team with an external facilitator to raise issues and perhaps one external architect for independent expertise.
- ALMA focuses on modifiability and again is driven by external experts called in to "certify" or audit the architecture at the end or middle, respectively, of the architecture design cycle [Lassing, *et al.*, 2000].

Eliciting stakeholder requirements ranges from "informally" polling to structured interviewing, and this is the prime input to the process. The evaluation itself is generally between architects and assessment experts and they take actions or communicate to stakeholders afterwards. However this is highly dependent on the phase in which the

[3] There are several reasons for this depending on one's attitude: (1) ensures an independent inspection (because there is no emotional bond with the architecture) and/or brings expertise in; (2) reassures management – "if outsiders think its OK then our people must be right"; (3) ensures work for architecture consultants!.

architecture is assessed; very early assessments are typically "architecture discovery" [Abowd, *et al.*, 1997] in nature, and thus a more intensive stakeholder-architect dialogue is warranted. In assessments held later in the architecture life cycle both family requirements and especially architectural choices are more established and the nature of the assessment is more *validation* that both entities are aligned.

In conclusion, the industrial assessments are still maturing and in ongoing development. With respect to information-system families SAAM may be considered to be one of the more advanced methods and a solid basis on which to base extensions to address the challenges identified in this domain.

4.4.4 Improvement opportunity

Based on the overview above: there is opportunity to extend current architecture assessment practice through:

1. Developing a generalisable "**how-to**" method that is geared towards architecture assessment in an information-system family context – especially in the important early phases of family definition when architecture is starting to take shape. This is crucial, as sustainable quality improvements are only possible when the **development team** assumes **ongoing** assessment responsibility and implementation embedded in their development process. Current methods, however (see [Abowd, *et al*. 1997]), stress dependence on external architectural consultants[4].;
2. Further extending the application of use-cases outside its conventional, functional focus to represent selected **family-relevant** (e.g. interoperability, extensibility), system-quality requirements;
3. Investigating the more **active incorporation of stakeholders** in the evaluation process (especially those strategic family stakeholders not addressed by traditional development methods, or SAAM) so that the architecture becomes more of a "living" organisation asset, and so that stakeholders can actively contribute to its evolution;
4. Developing more formalised, simple **techniques** and processes to support stakeholder-architect interaction especially as regards requirement *identification*, *specification* and architecture communication. The aim is to provide concrete advice on *how* to provide the deliverables needed for a SAAM-type approach.

These opportunities will be addressed in the FAAM (described in more detail in Chapter 5) assessment method to be developed in this research. FAAM has much in common with the existing SAAM-based methods; the key difference is the attention towards internally driven (see Figure 4.3) self-assessment of architecture by the family-development team as part of the ongoing creation and evolution of the architecture. This internal, improvement-oriented

[4] "Give a man a fish and you will feed him for a day, teach him how to fish and you will feed him forever" - Lao Tzu

approach (see Chapter 3) is contrasted with the majority of assessment methods reported previously; which are externally driven by experts, typically one-off occurrences per development project and consequently more certification and validation-oriented. The remainder of this chapter will elaborate requirements associated with the identified opportunities.

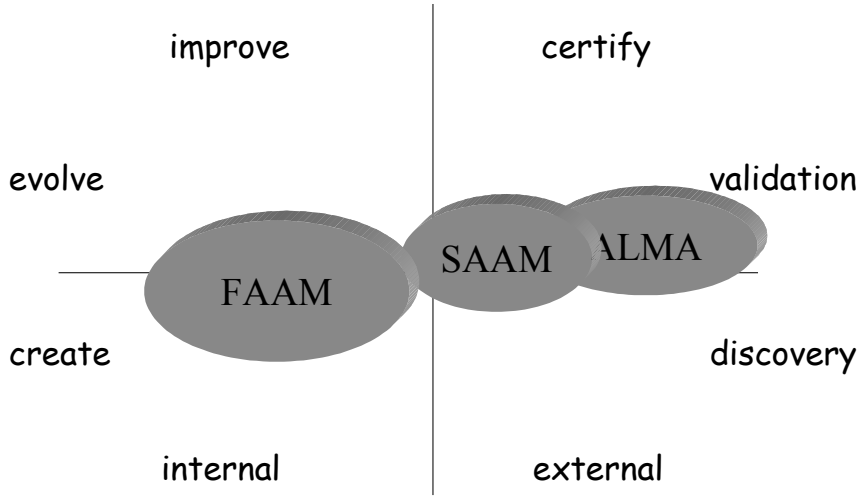


Figure 4.3: FAAM, a more improvement-oriented, internally driven assessment approach

4.5 Research scope (for operationalisation)

As has been mentioned repeatedly in this thesis; the domain of information-system family architecture is both complex and immature. In order to reduce the risk (and effort) in designing and demonstrating an assessment method for this domain it is essential to:

- scope the proposed work to a relevant, representative, subset of the domain representing an achievable goal considering the available research resources;
- base the method on appropriate existing stability, so as to leverage existing work and reduce the learning curve for industrial practitioners.

This section prescribes the context for the method development pursued in the research.

4.5.1 Research aims in context

It is important to clearly define the research goal with reference to the concepts of:

1. architecture development;
2. assessment methods.

This research regards architecture assessment as an important step in a larger architecture (and system) development process, see Figure 4.4. The research will *not* address a complete family-architecture development process, but concentrates on deriving an assessment method which supports (and is typically contained-in) such a development process. The assessment method is typically used to:

- examine an existing architecture prior to planning further change (this "ploughed field" is the starting point for much industrial family development);

- incrementally evaluate a candidate new architecture during its ongoing development.

A requirements-based perspective is taken and the assessment method recognises that a requirements process (to define requirements) and an architecture process (to define an architecture) co-exist in an overall software development process. These processes are themselves incremental in nature, and are interrelated (see Figure 4.4); the requirements serving as input to the architecture. The architecture assessment method proposed in this research is intended to strengthen this interrelationship and acts as an "interface" (for the selected system qualities) between the requirements and architecture processes.

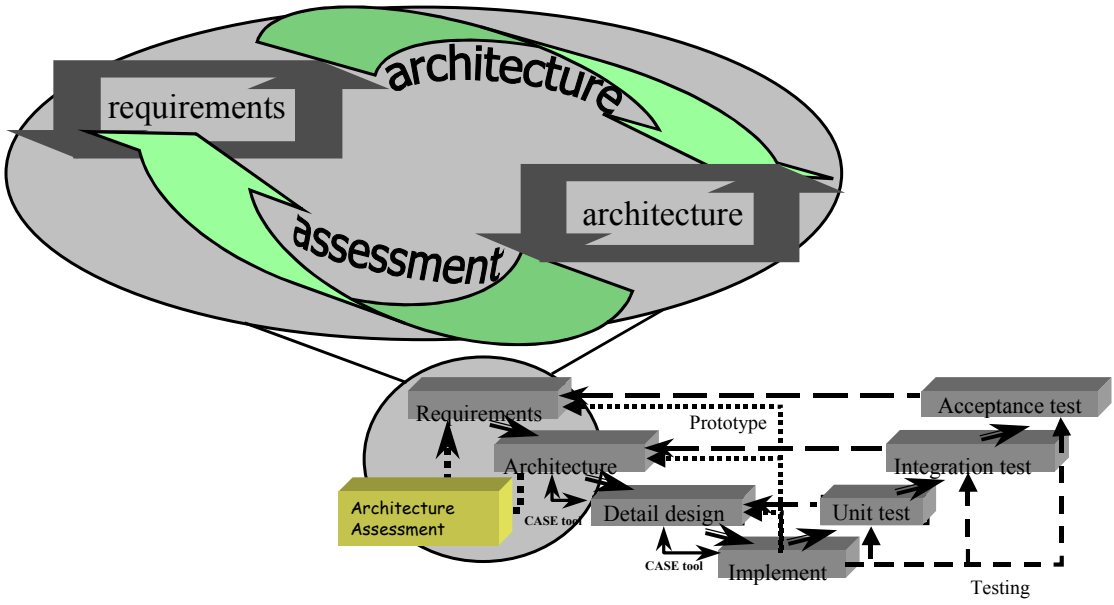


Figure 4.4: Assessment in family-architecture development cycle

Chapter 5 describes an assessment method for information-system family architecture and its *interface* within a larger, general process for information-system family development. As stated, it is recognised that architecture development takes place in a larger overall system (family) development process (recall the process and organisation aspect of the research framework illustrated in Figure 4.1) and assessment must be inserted as a logical step in this process.

With respect to interfacing the method in this larger context, the consequence for the research has been alluded to earlier when discussing the research framework. The assessment method must operate as seamlessly as possible within the tools, notations, and processes used in the overall development process. For this reason, reuse of best practices from current, popular development methods is favoured in the design of the assessment method proposed herein. Apart from these principles and the method guidelines for stakeholders, architectural notations, and method artefacts, no further details on method interfacing is specified.

Regarding assessment methods in general, the approach to assessment here is to improve the development process (improvement-oriented assessment); and is distinguished from

assessment for certification or customer acceptance as emphasised by assessment resources from standards organisations, e.g. ISO-9126. The differences between these approaches have been discussed in Chapter 3.

Information Box 4-A: What about the technical quality of the architecture?

Technology can play a strategic, enabling role in establishing markets and meeting business results. Such aspects fall under the *strategic*-needs dimension to the research framework in Figure 4.1. In the framework, the *technology* aspect is intended to convey how technical resources are used to address the strategic and operational requirements.

It is important to state that this research work does **not** assess the intrinsic **technical quality** of the architecture. This can be viewed as determining how well the architect used the available software resources (the *technology* aspect of the framework). Although an important part of the overall architecting process – it is not something the stakeholders are responsible for (or may necessarily appreciate). Having a "technically excellent" architecture is of less business importance than having one that satisfies the requirements.

Having a technically optimised architecture is, however, good practice and promotes the best use of technical resources. In order to ensure this aspect of overall architectural quality, a separate *technical evaluation* should be done. This evaluation is best done *after* the requirements-oriented assessment advocated earlier, because it is best suited to:

- a context of **stable requirements** – where the issue is less architecture *discovery* (what are the correct architecture goals for the system family) and more architecture-*efficiency*, (validating we have architected in the best way given we have addressed the correct problem?). In fact the rationale uncovered during a requirements-oriented assessment will be very useful in such technical architecture optimisation exercises. Conversely, any changes made for reasons of technical quality should be managed in the context of their impact on the strategic or operational requirements.
- clearly **focused issues** – the requirements-oriented assessment may be used to indicate architectural *hot spots*, which should be the focus for more detailed technical architect-peer-assessments. The prioritised requirements from the previous review also serve to provide a business-oriented weighting to such technical discussions.
- the use of architectural peer-review by **external experts** – technical reviews are the most logical use of external experts because technology is more generic than commercial application knowledge; and additionally companies may be very sensitive about sharing strategic *requirements* with outsiders.

Such technical assessments are not treated further herein.

4.5.2 Research focus

As mentioned previously the assessment method in this research will concentrate on the strategic aspects of information-system family architectures (see Figure 4.5) as indicated in the left arrow of the research framework. This aspect is associated with those non-traditional, strategy-oriented, stakeholders and system qualities relating to the long-term future proofing

of the family. Addressing this aspect is the first step in extending traditional product-based software development towards a more family-oriented practice.

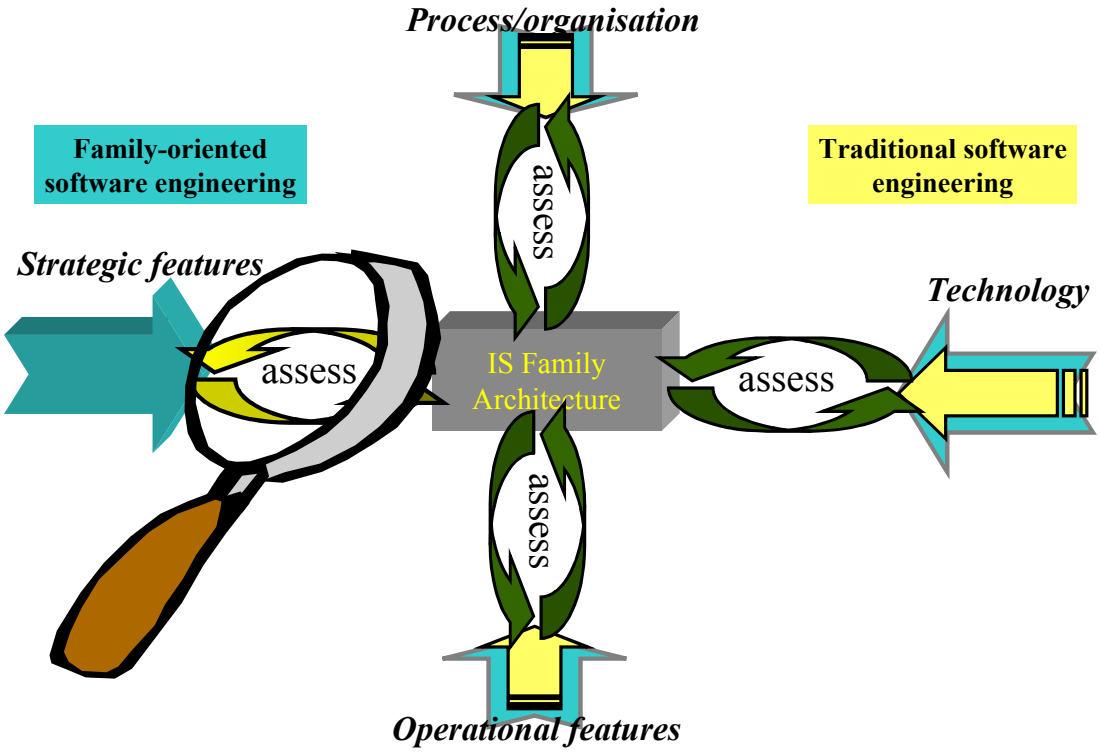


Figure 4.5: Research focus – the strategic aspects of information system family architecture

There are many system qualities important for information-system families and some have already been listed at various points throughout the text:

- *portability* (ability of the system to be developed and run on another execution platform using the same basic technology);
- *interoperability* (ability of the system to provide defined services through close interaction with external systems in the application domain);
- *scalability* (ability of the system to accommodate increasing workload and resources without decreasing performance);
- *extensibility* (ability of the system to accommodate extensions in functionality)
- *adaptability* (ability of the system to be configured so that its existing functions operate in a user and customer-defined way, this includes build-time and run-time aspects)
- *composability* (ability of the system modules to be reassembled, exchanged without having to do regression testing or modification on the whole system)**.**

This list is of course incomplete, all conceivable (single) system properties are also relevant to system families (see Chapter 3), and are all worthy of assessment depending on individual family circumstances and priorities. However, addressing all possible system qualities is not realistic in the context of a research thesis; so more focus is necessary in order to ensure that operationalisation of the proposed assessment method can be completed.

The research, thus, shall be limited to considering interoperability and extensibility, which are important system-family qualities considering the interrelated nature of the various products in an information-system family suite (see Chapter 3). They are especially important in supporting the needs for flexibility and integration typical of information systems. Addressing these qualities is an important contribution to the architecture assessment experience base; and will be used to operationalise the proposed assessment method. Their meaning and family relevance are explored in more detail in the next section.

4.5.3 Selected system qualities: Interoperability and Extensibility

Interoperability and extensibility have been chosen as qualities to operationalise the assessment method because they are very important to product family-based development, and are not yet well understood by architecture research or development methods. Both qualities are reviewed in this section and defined for the remainder of the research. Firstly, the treatment of the terms by the main quality standards institutes (ISO and IEEE) is reviewed; thereafter the definitions as used in this research are presented. The relevance of these qualities to information-system families, especially application suites, is reviewed and the extra, family, dimension to these qualities described.

4.5.3.1 Literature review

The ISO-9126 standard is dedicated to information technology quality. The aim of this standard (see [ISO-9126-1]) is the provision of definitions and metrics for software quality characteristics to aid specification of quality requirements and design goals.

Interoperability

- ***Interoperability*** is defined by the ISO as – "*the ability to interact with one and more specified systems (different to compatibility)"*.

This is a somewhat looser definition than that provided by the IEEE std. 610.12-1990: "*The ability of two or more systems or components to exchange information and to use the information that has been exchanged*". It regards interoperability as a sub-characteristic of functionality.

As regards metrics for interoperability, the basis of the ISO metrics is defined as "*the number of functions and transactions successfully transferred between the product and the connected systems*". These are then decomposed into simple ratio metrics based on comparing numbers of matched formats with total number to be matched:

- data exchangeability (data-format and user-success, based);
- inter-system standard consistency.

These metrics emphasise the *exchangeability* aspect of the IEEE definition but say nothing of the usage (semantic) aspect. This is somewhat understandable in the sense that the semantics are application-domain dependent, and therefore difficult to universally standardise. The problem remains, however, that such interoperability metrics specify a desired ratio which has no relation to the actual use of the systems and contains no valuable input regarding *what* interoperability should achieve in a functional context – which is after all the ultimate measure of interoperability. This lack of context has also been indicated [Kazman, *et al.*, 1996] as a reason for the weakness of such metrics for practice.

Extensibility

ISO does not address this issue directly, the closest it comes is in its discussion on *adaptability*, which ISO regards as a sub-characteristic of portability.

- ***Adaptability*** – "*ability to be adapted for different specified environments* ***without*** *applying actions other than those provided for this purpose to the software. (includes* ***scalability*** *of screens, reports, transaction volumes) NOTE: if software is to be adapted by the end-user then adaptability corresponds to suitability for individualisation as defined in ISO 9241-10, and may affect operability*." [ISO-9126-1]

The definition of adaptability as provided by ISO is closely related to that provided earlier in this chapter, and takes the standpoint that the software product *as is* shall deal with a changing application environment, typically by configuration. As before the metrics provided in [ISO-9126-2] are focused on simple observation-based ratios, providing overview information on the usage of the product. As an example – consider the adaptable data ratio "limit free operation after data adaptation":

- X=A/B where A = no. data operable after adaptation; B = total no. of data expected to be operable after adaptation.

With respect to *changing* the system – ISO also recognise a characteristic *changeability*:

- ***Changeability*** is defined as – "*the ability to enable a specified modification to be implemented (includes coding designing and documenting)*." [ISO-9126-1]

Modifications include corrections, improvements, and adaptations to changes in environment, requirements and functional specifications. The focus on changeability in ISO is on error correction and code-level changes.

This research project, however, identifies another approach towards dealing with increasing variety in the application domain – the "extension" of the *as is* software product with new functionality. This external (to the current system) approach reflects the reality that:

- extension of the system by third-parties will occur in information systems;
- customers and producers will want the ability to *extend* a system to a higher-class system in an easy and economical way.

These extensions will be treated more fully in section 4.5.3.2.

Based on the above review, the association between ISO changeability aspects and the more family-relevant characteristics of extensibility as indicated in this research is one where changeability can be regarded as a possible side-effect of extensibility. Normally, however, extensibility should be accommodated without significant code-level changes but by means of configuration and composition.

Conclusion
In conclusion, ISO-9126 provides solid definitions on quality and useful discussion on the notion of internal and external metrics and their relationship to the build-time and run-time (see [Bennett, 1997]) aspects of the system's life cycle. The focus of ISO seems to be on the *after-the-fact* assessment of software products, with summary metrics which provide a general indication of the state of the product. It is claimed that these metrics may also be used to specify requirements but this is insufficiently supported by illustration. As elsewhere, the link between metrics and appropriate corrective actions is not shown.

Further some of the issues regarded as important for this research are not touched-on by the standard (i.e. extensibility). Additionally there is no obvious family-focus in the standard, in fact there is an explicit statement that reuse and modularity (identified in this research project as underpinning family-oriented development) fall outside the standard. The metrics provided in ISO-9126 do not support requirements-based architecture assessment as envisaged in this research project.

4.5.3.2 Importance of interoperability and extensibility in information-system families

Interoperability is defined as:
The ability of two or more systems or components to exchange information and to use the information that has been exchanged (IEEE std. 610.12-1990).
In order to place a more business-oriented emphasis on the definition for use in the information-system context, it is extended here with:
...in order to support a set of coherent business processes that require the co-operation of the systems or components.

The following salient points can be inferred from the above definition:
1. There is a *syntactic* (format-based exchange) and *semantic* (usage-based exchange) aspect to interoperability.
2. Interoperability:
   - involves a chain of related activities (business process);
   - is expressed in terms of *multiple, co-operating* systems or components.

Interoperability – an information-system family perspective

Information-system families, with their attention to information integration, variety and life-cycle management, highlight the following interoperability issues:

- Information systems are typically part of a larger computerised value-chain and integration with other third-party systems is almost mandatory – therefore the family must interoperate with *external* systems;
- Family development aims to leverage investments through reusing components between different variant-systems. These reusable components must therefore interoperate with a range of other components both *internal* and *external* to the family. In Figure 4.6 below, those components representing: platform (C1, C2), modified-platform (C2'), and dedicated-line (C3) must interoperate with associated components so that they can be included and reused across lines.
- This interoperability is not only confined to developing a single software release from which various custom systems can be generated (e.g. sys. B.1.1 – sys.B.1.n in Figure 4.7 relating to extensibility, later). Information systems evolve and are upgraded (extended) across time through subsequent generations (e.g. sys.A.2 – sys.A.m in Figure 4.7), this means that systems and components must be interoperable *across* time. Interoperability, therefore, supports extensibility by ensuring that newly added components work with existing, older components.

Interoperability is a key quality in developing information-system families, not only in the conventional sense where the family member (system) must interact with its *external* environment – termed here *extra*-family interoperability. There is an additional dimension to interoperability in families, because of the fact that components within the family must interoperate with a variety of other *internal* components enabling reuse across family members, and across time. In addition, the product suite approach to families typical in industry [Dolan, *et al*., 2000], means that separate applications within the family (i.e. product lines, see Figure 4.6) must also present a high degree of interoperability to reinforce the family concept. A common example is the interoperability of OLE-objects across the MS Office™ suite.

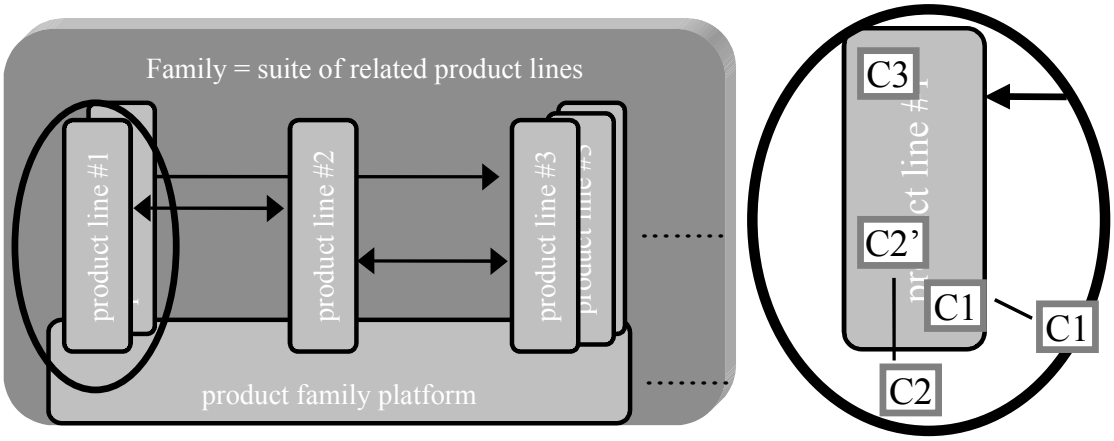


Figure 4.6: Family suite as collection of interoperable product lines

Furthermore, this high level of interoperability is used as a marketing discriminator to encourage customers to purchase additional members of the suite, see illustration in Information Box 4-B. The co-operation of components and systems within the boundary of the family is termed *intra*-family interoperability and is very important both in enabling reuse, supporting compatibility when components are (ex)changed to provide variety or extensions, and in advancing, to the customer, the added value of joining and remaining with the family.

Another important fact to remember is that while extra-family interoperability can be marketed as a bonus for customers (working with third parties is desired, but its absence can be excused by producers using claims that third parties are difficult to co-operate with because of competition); intra-family interoperability, however, is expected, and its absence is difficult to defend. Product-family producers are expected to have their product lines or components co-operating with each other, and failure to do so may raise questions of business credibility.

This thesis does not argue that new interoperability concepts are necessary for product families, but it does argue that a product-family approach makes interoperability much more important. Rather than being seen as having an external focus and "outside" the family it must *also* be regarded as being a core-competence necessary for the "internal" working of the family.

This is not to say that extra-family interoperability can be ignored and all resources concentrated on intra-family interoperability. Interoperability, with is attention to the boundary of the system, is an important architectural aspect. It is concerned with setting and accommodating the system in its larger context (the macro-architecture, [Lassing, *et al*., 1999]). This is especially relevant in information-system families, where integration of data and functionality across business-processes, enterprises and system families is increasing. Changes from the external environment impacting interoperability are to be expected and an exclusively inward-looking focus is not a clever strategy. The provision of interoperability to third parties is also important for maximising family returns, as it allows one to:

- sell family members as parts of larger multi-vendor composite systems;
- trade finer-grained components to and from other vendors.

This also motivates the trend to use open-standard interfacing (e.g. CORBA, XML) between internal family members and components. Family-based development of information systems means that more attention will have to be devoted to both extra- and intra-family interoperability.

Information Box 4-B: intra and extra family interoperability and extensisbility – examples

Members of the Philips Matchline™ family in home entertainment systems must naturally support conventional *extra*-family interoperability using industry standards to operate with multi-vendor video and broadcast services and systems, e.g. a Philips TV must interoperate with a Sony VCR.

In order to advance the value proposition to the customer of remaining or expanding his investment in the matchline™ family; additional interoperability features between (intra) family members are offered. For example the ability to easily synchronise clock-settings i.e. make the setting change once on a single system (e.g. video) and have it propagated automatically to the other family members (e.g. TV, stereo, video camera). This leveraging of internal company/family knowledge and cohesion to provide more integrated services across family members is a means to discriminate family members from other systems on the market.

Similar illustrations can be found in the case of extensibility. In the case of medical information systems customers deploying such systems typically do so in a phased process, gradually incorporating more of their operational business support into the system. This can be a long (years) process and must be aligned with their budgeting plans. They expect a growth path between the systems in the family (*intra*-family extensibility) so that their gradual approach to incorporating functionality and providing expenditure is supported with specific fielded systems in different price and function categories rather than promises of things to come. For example radiology customers may begin automating their image production and distribution operations using PACS systems and will later expand the automation effort to include their administrative and reporting activities using a RIS system. The customer expects the producer to have systems dedicated for these function-categories, and expects that they can easily extend from the PACS-only to the PACS and RIS category.

In terms of *extra*-family extensibility, medical customers would also expect that the PACS and RIS systems would have (API) interfaces defined allowing them to integrate best-of-breed applications from third-parties, e.g. speech-recognition systems to support reporting.

Extensibility – an information-system family perspective

In a family context a distinction is made between customisation and extensibility as important means of providing market flexibility. Customisation is defined here as the ability to adjust *existing* functionality to suit individual customer or user wishes. Extensibility, on the other hand, deals with the provision of new functionality.

Extensibility is defined in this research as:
The ability of the system to accommodate the addition of ***new*** *functionality*.

There are two obvious means of offering new functionality:

1. The producer offers new features as upgrades for which he is responsible as part of the value contribution to the customer. The producer is responsible for the correctness and continuity of the feature and has control and complete insight into it. The extensions are configuration items managed by the producer. Such extensions are termed *intra*-family extensions.
2. The customer develops his own additional features using infrastructure provided by the producer (e.g. scripting language, API). In this case the customer is responsible for the correctness of the extended feature, while the producer is responsible for maintaining the continuity of the infrastructure. This capability to support *extra*-family extensions

provides most freedom to the customer but must be managed carefully so that the producer does not end up supporting or repairing customer extensions. On the positive side it is a good way to capture market needs and let customers produce features which may be later adopted as managed (intra-family) extensions to the family, i.e. "growing the family". The open-source movement and LINUX are extreme examples of this approach (see e.g. [Raymond, 1998]).

Offering future extensions is a common concept in the entire software industry, most readily manifest in the fact that successive releases or versions of systems usually include extra features (see the full-line progression between sys.A1 – sys.A2 in Figure 4.7), often determined on a single-version horizon.

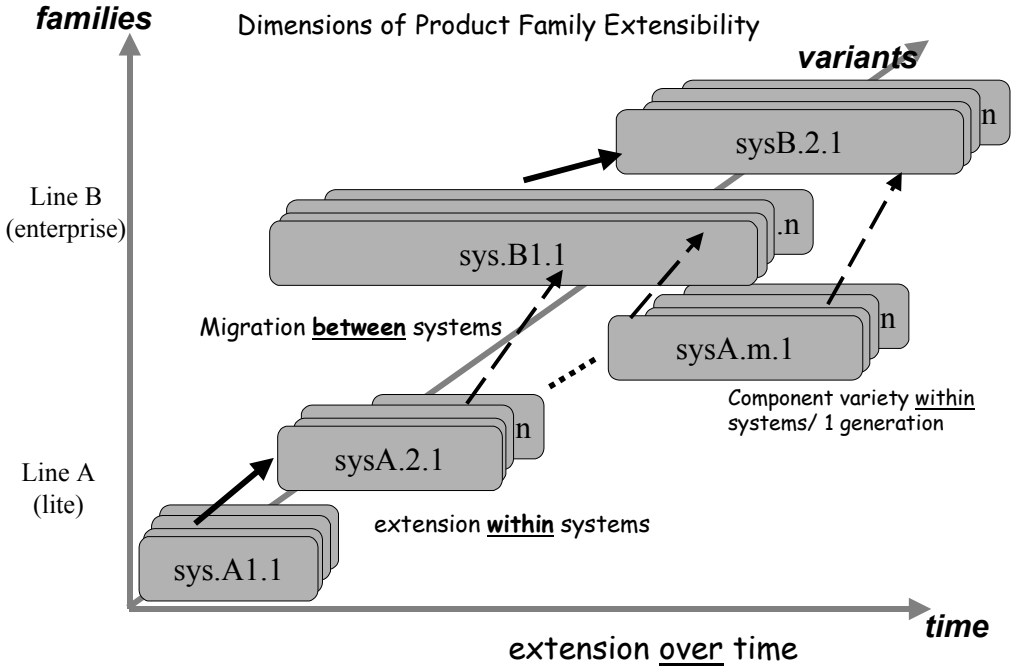


Figure 4.7: Extensibility (intra-family) in and across time

A particular aspect of extensibility in relation to families is the differentiation of family members or variants based on producer-managed features. An obvious (non-software) example is the different types of credit and frequent-flyer cards offered by individual companies. The producer offers different types of product, with increasing levels of service (and cost), but allows the customer to migrate between them thus offering him a means to extend his product within the family. This is particularly effective in providing a low threshold to entering the family by means of a basic product type, which attracts the typically large, low-end market segment. Once this customer base has been captured, extension can be used to manage their migration to more advanced, and more expensive, family members. This approach also has advantages for the customer as it allows him to stagger his investment and

business reorganisation by means of a gradual adoption of increasingly powerful support, from within the context of a managed product family.

Of course in information systems there are some consequences to providing extensibility. Customers may want to buy entry-level ("lite") system, and then later migrate (or extend their system) to a higher-class system from the family (see the dashed-line progression between sys.A2.1 – sys.B1.1 in Figure 4.7). However, the uplift of the low-end system must proceed seamlessly and preferably cheaply (i.e. reusing data, information structures and, where possible, other software and hardware infrastructure-elements). It is important therefore that the features to be migrated must be understood. The figure above is more than a project release schedule, the content of the variants and systems should be described and their interdependence across time made explicit.

This ability to extend systems serving one market segment to another is especially important towards maximising reuse to realise the economies necessary for families. As with interoperability, extensibility provides an important means to evolve and maintain the family, especially as a means to guide the evolution of customer functionality (and hence revenue streams). This ability for the customer to phase his investment along an explicit roadmap is an illustration both of the strategic dimension to extensibility inherent in information-system families and the fact that the customer is aware of the explicit existence of the family. These important "ilities", their specification and assessment, will be addressed in the remainder of this research.

4.5.4 Selected application domain

The pervasive aspect of information systems has been presented in Chapter 1. The field is very broad and addressing the interoperability and extensibility of all information systems is a risk-laden challenge. In the interests of practicality a single area of information systems will be addressed in the research – enterprise information systems (EIS). This category is very representative of the general community (earliest commercial information systems started here) and is the "archetypal" information system. In particular, the characteristics of information systems discussed in Chapter 3 apply to this class.

For reasons of operationalisation and evaluation of the method, two examples of such systems: a medical radiology EIS, and an enterprise planning EIS, will be assessed. These systems are used for supporting the primary operational processes and management reporting in both the radiology department, and large manufacturing or assembly companies, respectively. More details of the evaluation environment will be provided in Chapter 6.

Now that the basis underlying the proposed assessment method (SAAM) has been described, and the research scope has been focused; the remainder of this chapter will concentrate on addressing the first research question posed originally in Chapter 2. This relates to the

stakeholders and the representations involved in the assessment. After considering the participants and the representations used in the assessments, some implementation requirements constraining the remaining method design are presented. The design itself is presented in Chapter 5, and its implementation in Chapter 6.

4.6 Stakeholders in IS-family architecture assessment

This section concentrates on the first research question to be addressed in this thesis:

1. **How can the primary stakeholders of information-system families be identified, and their quality requirements captured?**

4.6.1 Stakeholder profile

Taking a stakeholder-oriented approach to development has been emphasised throughout this research as important for families, and has also been adopted by the emerging software-development methods (e.g. RUP from Rational™). Similarly in developing the assessment method – a stakeholder analysis will be done to determine the characteristics and concerns of information-system family stakeholders (i.e. the users of the method).

4.6.1.1 Stakeholders in the research framework

One of the most noticeable features of the research framework (see Figure 4.5 earlier) concerns the separation of requirements across strategic and operational aspects. In the "operational features" aspect we find the *traditional* stakeholders of the system. These stakeholders are representative of those people typically involved in most (single) product-development projects: e.g. user, architect, marketer, developer, tester, purchaser, and maintainer. These system stakeholders (i.e. the system users and the various development stakeholders) tend to have a single-system focus, and have been the "target audience" of conventional requirements engineering, and development methods. They concentrate on addressing the operational requirements of the product in "run-time" (see [Bennett, 1997] and Chapter 3).

An underlying (if non-explicit) assumption of such development environments is that the *customer* and *user* are equivalent or at least have common aims, mainly concentrating on the so-called *functional requirements* of the system. A characteristic of conventional product development is the fact that all other features of the system are grouped under the heading of *system qualities*, and they receive less-than-optimal support from the various development methods.

There is, however, often a significant difference between customer and user. The customer is the party who pays for the system, while the user is the party who actually interacts with the system. There may be overlap in concerns, however, and in certain cases the customer activates some system functionality. In general, however, users are *primarily* concerned with

functions, performance, and user-interface aspects of the system; while customers are *primarily* concerned with so-called system-quality aspects such as: reliability, extensibility, openness, maintainability, migration, compatibility. In the consumer-goods (e.g. shavers, televisions) market it is usual that the distinction between user and customer is artificial; in professional systems (e.g. medical systems), however, the parties buying (e.g. hospital financial controller or information-system manager) and using (e.g. radiologist) the product can have different, and sometimes even conflicting priorities (see e.g. [Macaulay, 1996]).

The research framework recognises this dichotomy and aims to reduce the complexity of user or customer requirements for development by regarding them separately[5]. Strategic features such as reuse, portability, and openness are the concern of strategic stakeholders e.g. business manager (producer), customer, development manager. Operational features concerning functionality, performance, and usability of the system are the concern of e.g. users and service technicians.

An important result of the difference between user and customer is that many of the non-user issues, which in the past were left to the firm to solve, have now become the explicit concern of the increasingly influential customer. Many modern customers now want the freedom to stagger their investment over time so as to better manage their cash-flow; so they demand extensible products where they can buy add-ons according to their own financial schedule. Similarly they demand open products so that they can exercise their power in the buyers market by choosing different vendors to satisfy different aspects of their total requirements, yet ensuring interoperability. More detail on these strategic-level customer requirements will be provided in a later section addressing all stakeholder needs.

The firm developing the system family, the producer, must account for these strategic customer needs; but also has strategic needs of its own related to effective product-family management. In order to guarantee the best return on investment, through maximising the use of common features, the firm has to proactively address reuse across all levels in the family. As another example, in order to achieve maximum market penetration or to benefit from platform advances the firm may also have to ensure that system portability is addressed throughout development.

Such strategic-level, (i.e. spanning multiple-development-projects, and products) issues affecting customers and producers are explicitly treated in this research framework, and assessing how well the software-system architectures address such issues is the main goal of this work. An overview of the stakeholders in information-system family development, with particular emphasis on the strategic stakeholders, follows.

[5] As an aside, the separation of concerns above suggests a more natural division of requirements into *user* and *customer* as distinct from *functional* and *non-functional* respectively.

4.6.1.2 A survey of information-system family stakeholders

Who are the stakeholders in software-system families?, and how do they interact with the architecture? Non-trivial questions given that architecture is pervasive throughout the development process (see e.g. [Hammer, 1996], [Bennett, 1997]). This section will identify the major software-system family stakeholders; and their association with the important family qualities of interoperability and extensibility identified earlier. The method used is based on literature review, and practical observation from industry. In order to keep the results as general as possible, no pre-conceived family-development method has been used. However, as in all things organisational (as product families clearly are!) some peculiarities in approach and terminology will inevitably creep in, and readers are encouraged to actively edit labels to suit their own context.

The [IEEE 1471, 2000] definition of ***stakeholder*** as: "*an individual, team or organisation (or classes thereof) with interests in or concerns relative to, a system*"; has been introduced in Chapter 1 and will be used throughout this work.

It is important to recognise the fact that in market-led organisations, not all possible stakeholders can be directly involved (e.g. all potential users) so there must be *internal*-representatives for many actual stakeholders, not only for those entities involved in using the system, but also for those building the system. This of course is also true for *other* systems which use the software system (e.g. salary-systems coupled to time-registration systems) – and whose stake must be represented by people.

Software-system stakeholders may be categorised by their "stake" in the system. Table 4.2 below identifies four categories of development stakeholders (based on [Macaulay, 1993]), and gives examples of possible stakeholders (and their roles in architecture) for each (from [Gacek, *et al*., 1995]).

Table 4.2: Stakeholder roles – initial adapted from [Gacek, *et al*. 1995]

Stake (abbreviation)	Stakeholder	Roles/concerns
Financial (F)	Customer	• schedule and budget tracking • risk assessment • requirements traceability
Development (D)	Developer	• complete consistent architecture • requirements traceability • support for trade-off analysis • select/develop s/w components • maintain compatibility with existing systems
Support/customisation (S)	Maintainer	• maintain compatibility with existing systems • receive guidance on s/w modification and family evolution
Usage (U)	User	• performance, reliability, compatibility, usability, • accommodate (future) user/legal requirements

The strong emphasis on requirements traceability, compatibility with *existing* systems, and accommodation of future needs is indicative of Gacek's rationale-based approach to architecture. This contribution has proven useful in identifying software-system stakeholders; and the explicit identification of roles is important and more revealing than a simple check-box. A feature of this, and indeed much other research in software architecture, is the emphasis on single-system development.

But reality bites! The approach towards identifying software-system family stakeholders and their roles must reflect the reality that family-oriented research is relatively sparse, and that most work is based on non-family environments. Furthermore, most software companies (in reality) realise families through combining related, but distinct, pre-existing, product lines that have been developed as single-products (see [Jacobson, *et al*., 1997]). So most stakeholders experience development of families as a set of related product releases within a strategic family framework. For these reasons, the approach for the remainder of the paper will be to present the stakeholders and their roles, concerns from both a conventional product development perspective and from the perspective of family development. This will provide stakeholders with a link to their recent "single-product reality" while also clearly indicating those parts of their job related to family issues. The idea is to present a comprehensive list of stakeholders and their concerns – this will serve as a trigger to:

- identify stakeholders across companies based on their concerns (rather than labels, which may change);
- prompt stakeholders as to what issues they should be considering, and hence generate requirements for assessment;
- identify the areas needing attention when designing practical support (techniques) for stakeholders in the proposed method.

The list in Table 4.3 below is an extension to Table 4.2 to address the family-aspects, and is based on: the characteristics of software-system families presented in Chapter 3; the collected experience of the authors; and various contributions from literature, notably [Hammer, 1996] and [Jacobson, *et al.,* 1992]. In the interest of brevity the list will provide keywords to identify roles. Rather than presenting an exhaustive set of arguments for all stakeholders and roles[6], this section will confine itself to abstracting the general method from that rationale for a specific stakeholder, to be provided later. Shading is used to indicate a link to the original stakeholders identified in Table 4.2.

[6] This is reported in an internal research-project report, any specific cases may be discussed via correspondence with the authors.

Table 4.3: Software-system Family Stakeholder roles

Stakeholder (S take)	Product roles/concern	Family roles/concerns
1. Customer (F)	• schedule and budget tracking	• strategic alliance • accommodate future requirements • flexibility (compatibility, scalability)
2. Customer (D)	• risk assessment • requirements traceability	• flexibility (interoperability, customisation, extensibility, industry-standards, state-of-the-art technology)
3. Customer (S)	• reliability • maintainability	• upgradability • use of industry-standards
4. User (U)	• performance, reliability, interoperability, usability	• consistent features across family (e.g. UI) • increasing levels of performance and usability • correspondence between system and application domain
5. Business management (F)	• track individual product relative to policy/targets/roadmap	• Overall family **business policy/targets** • market segment • release policy/business roadmap • make-buy (outsourcing)decisions • strategic alliance management
6. Product management (F)	• derive individual product commercial targets; • product-content/priorities	• derive product roadmaps • defines commercial options/features • commercial configurations
7. Marketing/Sales management(F)	• **represents Customer(F)** • individual product market/sales plan • product compatibilities and configurations	• **represents Customer(F)** • market family aspect • returns market info. on preferred options • must know current/future configurations and compatibilities
8. Customer/Operations support management (S)	• **Represents Customer(S)** • Guidance on software modification. • compatibility with existing systems. • product delivery, maintenance and customisation • testing	• **represents Customer(S)** • maintain compatibility with existing systems across upgrades • receive guidance on s/w modification and family evolution
9. Application specialism (U)	• **represents user (U)** • applicational integrity of product • defines user functions, use-cases • beta-testing • user-training	• **represents user (U)** • applicational integrity of family • reviews application domain model • indicates future requirements • context of family in user-environment • user-based options features/configurations
10. Development management (D)	• **represents Customer(D)** • ensure design of product within family constraints • co-ordinate inter-project resources • provide development infrastructure • development personnel management	• **represents Customer(D)** • provide technology roadmap supporting business roadmap • provide for current/future requirements • satisfy flexibility and reuse requirements (rationale/assessment) • reference architecture for family evolution • technology watching to influence business roadmap • matches development capabilities to family business strategy • state-of-the-art tools and techniques • evolution of development process. • Standardises/reuses cpts./practices across families**.**
11. Development project Management (D)	• project schedule/budget/resource/quality • feedback to product/business management	

Stakeholder (S take)	Product roles/concern	Family roles/concerns
	• formal project process	
12. Requirements analysis (D)	• analysis of stakeholder requirements for product • requirements specification/management/traceability	• application domain modelling • architecture assessment • analyse new requirements in family context
13. S/W design (D)	• select/develop/test s/w components to specs. • maintain compatibility with existing systems	• use state-of-the-art technology • balance of new and reused functions • communicate technical issues to business/technical management
14. Purchasing Management (F)	• co-ordinates project component/services supply	• assesses development partners • provide ideas to reuse/standardise purchases across families

The table above is very detailed, and the result of much research and discussion. In order to provide an idea of the general process involved, the customer stakeholder (previously identified as very important in software-system families) will be examined in some detail here. The most obvious feature we see is that the customer is very prevalent throughout the list; how did this happen? The various stakes provided by Macaulay, [1993] were gathered; these stakes were associated with selected keywords and stakeholders (if mentioned) from Chapter 3. The fact that the customer has a financial stake (Customer (F)) is obvious. Chapter 3, however, talks about "future-proof extensions at the customer site independent of the producer"; this is essentially development under the responsibility of the customer – so a development stake has been identified for the customer (Customer (D)). Similarly it was stated that customers wanted individual customisations preserved across upgrades; concern with compatibility across upgrades is normally the task of support; hence Customer (S). This process is repeated until the candidate stakeholders have been covered. Hereafter remaining keywords are associated with new (to Macaulay) stakeholders typical from market-led development organisations.

Some of the most relevant issues from Table 4.3 are highlighted below:

- Almost all stakeholders have a role to play in family development, relating the roles here to appropriate architectural views and discovering how these views shall best support the role is the challenge to software-system family architecture.
- Internal stakeholders which represent the multiple external stakeholders (shaded grey) of market-oriented development are indicated e.g. application specialists represent the multiple users of the system.
- The key issues of software-system families raised in sections 2 and 3 (e.g. flexibilities, strategic planning, requirements traceability) have been allocated to stakeholders.
- A clear distinction between family and non-family activities and stakeholders (e.g. Development Project Management has no role in family) has been made.
- The architect is not included in the list of stakeholders – the architect's role is to develop a compromise development framework satisfying the combined concerns of all the stakeholders. As will be shown later, those actually doing architecting will often assume

the role of another stakeholder during development, but then they are not (playing) the architect.

The list of stakeholders and roles is long, and should be edited by readers to suit their own organisational context. The next section will provide such an example editing which maps the list to a more realistic organisational setting, indicating a generic combination of stakeholder roles for family management.

4.6.1.3 Software-system family stakeholders – a practice-oriented reduction

The number of stakeholders listed in Table 4.3 is relatively large[7] even considering the internal stakeholders who replace the customers and users (and could even be extended depending on individual organisation structures). This is particularly so when one considers the practical concerns of running effective review meetings and structuring task forces or management-teams in an industrial context. This section presents a re-allocation of *internal* stakeholder-roles to reflect the fact that individuals in *family*-management may fulfil multiple stakeholder-activities in a real organisation. These re-allocations are intended to:

- ease the mapping to common business management-team structures;
- be useful guidelines – not rigid rules.

The following combinations of stakeholders have been made from Table 4.3.

Table 4.4: practical reallocation software-system family stakeholder roles

Stakeholder (S take)	Product roles/concern	Additional Family roles/concerns
1.Business management (F)	•as in Table 4.3	•as in Table 4.3 •represents Purchasing management(F)
2.Product management (F)	•as in Table 4.3	•as in Table 4.3 •represents marketing and sales management (F) •represents Customer (F) •represents Application specialism (U)
3.Customer/operations support Management (S)	•as in Table 4.3	•as in Table 4.3 •represents Customer (S)
4.Development Management (D)	•as in Table 4.3	•as in Table 4.3 •represents Customer (D) •represents Requirements analysis (D) •represents s/w design (D)

The above re-allocation has mapped all internal and external stakeholder family roles from Table 4.3 onto the 4 key-business stakeholders: business management, product management, customer support management and development management. This is *a* realistic organisation of family responsibilities considering industrial family-management structures – there are of course others and individual organisations will establish their own mapping. As stated previously, regardless of the number of people involved, the important point is that the

[7] The list is also considered long in deference to George Miller's [Miller, 1956] "7 +/- 2" theory on the storage limit of short-term human memory.

concerns and responsibilities of all stakeholders are addressed by the product-family management team. The message of Table 4.4 is that the key family stakeholders listed must possess, or have access to, the skills of those other stakeholders they represent at the family-management table. Another important issues is that the various activities and responsibilities are *clearly allocated* to named individuals and that this allocation is communicated throughout the organisation.

While product families present challenges to the producer, the consequences for architecture are, in general, not new (see Information Box 4-C). Previously existing (single) system qualities have been emphasised by the demands of family development [Van den Hamer, *et al.,* 1998]. This is actually an advantage – because it means stakeholders are somewhat pre-disposed to the concepts, even though support in current practice is minimal [Hammer, 1996].

Information Box 4-C: No new stakeholders?

Some researchers have recognised that not all is the same, and have expressly emphasised the differences in a family context. They have brought some family-specialisation into the stakeholders presented above. As an example: the following family-specific stakeholders are identified by Clements and De la Puente, [1998]:

- Product line architect;
- Generic asset builder
- Product builder (from generic assets)
- Product line maintainer
- Product line marketer/funder

Examination of these stakeholders reveals, at least in some cases, that the word product-line (family) has merely been prefixed to the traditional stakeholder (architect, funder, and maintainer). In a product-family organisation these (un-prefixed) roles are by definition product family focused (typically you're making product families or you're not) and are implied in the previous product-family-focused discussion above. In other cases, such as the asset builder and product builder, there is a difference in focus between component development and component integration. At this stage in the research this internal-development stake is not considered. Furthermore, the implied organisational division between asset-building and product-building, is not universal in practice (see [Bosch, 2000]).

4.6.1.4 Information-system family stakeholders in this research (Research question 1)

The first research question is generally answered in this section – the main categories of product-family stakeholders (based on the previous mapping) and their (interoperability-, extensibility-related) concerns are identified and architecture requirements are derived. The main stakeholders, and the related secondary stakeholders, can be identified based on the reduced set presented in Table 4.4 above. Table 4.5 below provides a summary of the main stakeholder tasks, which can be used to help identify them and their concerns.

Note: The organisational-mapping contained in the table below reflects those roles involved in practical family management in an organisational context. However, this does not strictly

conform to stakeholder theory – a system's stakeholders are *all* those involved with the system; therefore customer, architect, user are all valid separate stakeholders (as indicated in Table 4.3). In order to keep the concept manageable, and to reflect that a limited number of people are responsible for the family, the subset of stakeholders below will be maintained. It must be remembered, however, that these internal stakeholders must represent their own interests, those of their staff, and those of the external customer, user (person or machine), or government – as also indicated previously in Table 4.3.

Table 4.5: Stakeholders and concerns for interoperability, extensibility

Stakeholder	Concerns	Architecture req.'s
(Family) business management	• Strategic alliance with customers/partners • Overall family strategic roadmap (and role of members therein) • Align family with market • Manage commercial configurations • Manage make/buy policy	• Support business strategy • indicate make/buy flexibility • support for market segmentation • support commercial configurability • support commercial differences/similarities
Product (family-member) management	• Individual family-member roadmaps in family context • Define family-member functional and *quality* features • Must know current/future customer configurations and requirements (interoperability within application domain)	• Support system qualities • Show openness/extensibility of family member • Show interoperability with domain • Show ability to accommodate future customer/user requirements
Customer support management	• Promote continuity and installability for customer	• Support customer migration (extension) • Support seamless upgrade according to customer requirements
Development management	• Responsible for technical roadmap underlying family (architecture) • Aim for reuse and flexibility (e.g. interoperability, scalability, etc.) between components within and across family members	• Show make/buy components and their relationship • Align technical strategy with development organisation • Show use of standards to guarantee interoperability and future-proofing.

Summarising – the above stakeholders are primarily concerned with strategic aspects of the family (left arrow of the research framework) – concept such as "future" and "roadmap" are common. These concepts have been instantiated in the architectural requirements in column 3 of Table 4.5 above. A very important observation is that the stakeholders themselves must be clear on their requirements so that the architecture can address them. Clarifying the system qualities for these stakeholders is a focus-point for the assessment to be developed in this research.

In that respect requirements and architecture representation will play a crucial role in the assessment. This issue is explored in the following section.

4.6.2 Requirements and architecture representation

Representations have already been discussed in Chapter 3 when dealing with assessment techniques. This section will present the highlights of representation-related discussions from throughout the thesis. The position in this research is to leverage emerging best practice with respect to representation rather than inventing (yet another) notation. This is motivated primarily by the observation that high-level architecture assessments involving stakeholders generally use simple notations, the types and granularity of which is determined by the particular use-case and stakeholder context (see [Bass, *et al*., 1998] pp 218). Furthermore, reviews are primarily vehicles for focused discussion, and any representation which helps exchange information is sufficient (see [Bass, *et al*., 1998] pp233).

4.6.2.1 Requirements representations – the change-case

System qualities, the focus of this research, are typically not explicitly represented in contemporary software development methods. Representation efforts in the architecture assessment methods (discussed earlier) concentrate on architecture but requirements representation is typically of the "one-line sentence" variety. This is to be expected as the majority of existing architecture assessment method are driven by parties external to the development team, and the level of requirements detail known to them is aggregate.

The approach in the proposed FAAM method advocated in this research is to enable organisations to self-assess themselves, with the direct and more active participation of stakeholders and their more-detailed knowledge. The proposed assessment method is use-case based so representations from this field will be leveraged as much as practical, especially as this technique is oriented to stakeholder understanding [Jacobson, *et al*., 1992]. This also follows general practice, where system qualities are represented by textual descriptions and this is advised as sufficient (e.g. [Bennett, 1997] pp68). This textual use-case basis will be extended with the concept of change-case introduced below.

As noted during the SAAM description (4.4.1) earlier and in Chapter 3 many of the system qualities are concerned with how the system can accommodate changes in its internal or

external environment. The natural association between system qualities and change is emphasised in this research, with the decision to represent the system quality requirements in change-cases. The term ***change-case*** (see [Bennett, 1997], pp63) is used to describe those candidate changes that must be accommodated by the system. They play the same role in representing requirements for the "build-time" aspects of a system, as the use-cases play in describing the functional requirements associated with the "run-time" operation of the system. The new term distinguishes these two aspects of the system's life-cycle, and emphasises the fact that change-cases specifically deal with issues of most interest to the strategic system stakeholders (e.g. producer and customer), in particular the future modifications to the system and the system's relationship with others. These are among the strategic product aspects of the research framework indicated in Figure 4.1.

It has been observed in Chapter 3 that roadmaps and variety are important concerns in product-family development; and there is, therefore, a strong requirement to represent the variety and temporal aspects of families in such roadmaps. Regarding the representation of families – current dedicated family representations (e.g. GBOM, see [Hegge, 1995]) are focused on the operational phases of the life cycle. In the design phase of families conventional software design methods and associated tools (e.g. UML) are being applied [Bass, *et al*., 1999] but are not specifically developed for family representation. The method aims to leverage emerging developments in good practice with respect to family representation but will not devote effort to developing a family notation. The change-case concept will be extended to address family aspects and to provide concrete guidelines towards helping the stakeholders to generate and structure their requirements for the particular qualities of interoperability and extensibility. Such extensions are developed in the method design, described in the following chapter.

4.6.2.2 Architecture representations

As mentioned in the architecture survey in Chapter 3, there are almost as many architecture representations as definitions. In this thesis the experience of SAAM/SEI (see [Bass, *et. al,* 1998]) with respect to mandating a specific representation is incorporated – i.e. don't. As stated in SAAM [Kazman, *et al*., 1996] – any existing stakeholder-familiar representation should be incorporated where it useful. However, where existing representation is absent the use of Kruchten's [1995] 4 + 1 views and the UML notation [Rumbaugh, *et al.*, 1999], [Fowler and Scott, 1997] is *advised* as a pragmatic position due to the increasing popularity of these approaches in the practising community [Bass, *et al*., 1999], and (in the case of UML) the fact that the drawbacks are outweighed by the advantage of a standard notation [Hofmeister, *et al*., 2000] pp. 16.

The particular architecture views (see Figure 4.8) used during an assessment depends on the system qualities under investigation. For example [Kazman, *et al*., 2000]:

- the 4+1 *logical* and *development* views is useful for reasoning about the *build-time* [Bennett, 1997] quality of modifiability;
- the 4+1 *process* and *deployment* views are useful to assess *run-time* [Bennett, 1997] concerns such as performance and availability.

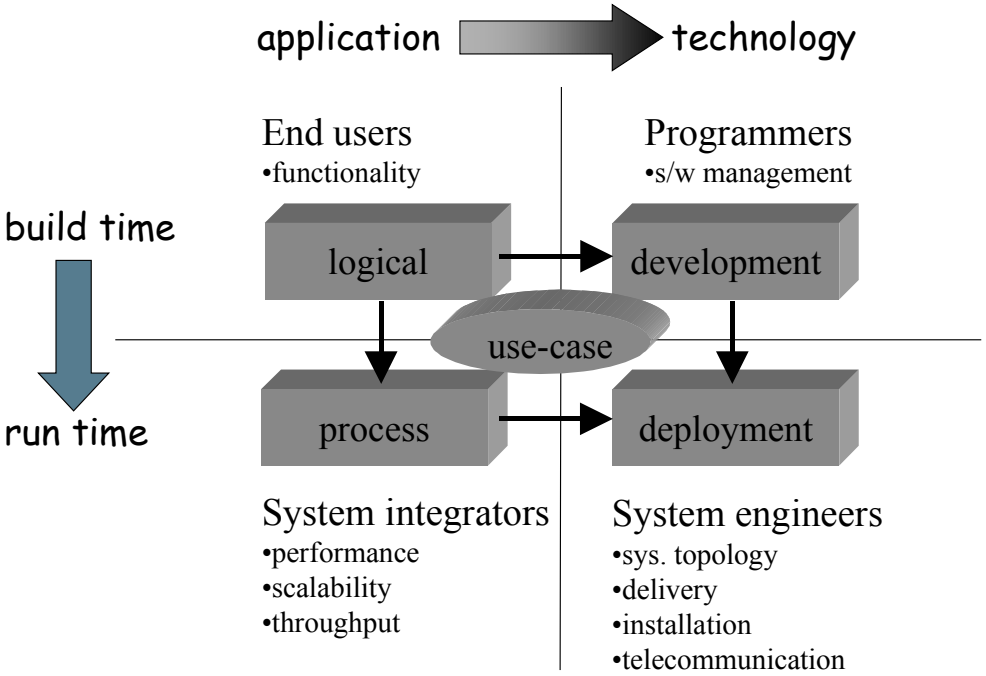


Figure 4.8: 4 + 1 views of system architecture (extended from Kruchten [1995])

The views of the 4+1 framework are regarded in this research as addressing the important aspects of the architecture problem space. Figure 4.8 above relates the original views to the build-time and run-time aspects of the life cycle observed by Bennett [1997], and to the application and technology concerns at the heart of architecture.

The logical and development views, with their attention to domain concepts and organisation of software resources in the developing organisation, respectively, are most closely associated with build-time issues, the definition and management of the software by the producer. Similarly, the process and deployment views are most closely associated with the dynamic behaviour and topological organisation of the system in its application environment i.e. the run-time aspects of the system.

Regarding the application–technology divide, a sliding scale is more appropriate than a sharp division. In general, the logical and process views are concerned with the static and dynamic aspects of the application-functionality from a usage perspective e.g. performance and scalability. The development and deployment views are most concerned with the technology-oriented organisation of the software in the development environment, and the allocation of the system across technical resources in its usage environment.

This categorisation of views and perspectives are a guideline to indicate that although *all* views are important for the complete development, deployment and maintenance of a system family, not all architecture views are needed to explore a single perspective of a single stakeholder on a single quality. Furthermore, architecture assessment should occur early in the family development cycle, and not all views will have been completed to the same level of detail. While all views evolve concurrently as part of an incremental development cycle, following the influences indicated by Kruchten, the relative refinement of views are context dependent. Initial deployment decisions, for example, will be very aggregated in the information-system domain compared to the real-time embedded-system domain[8] where performance concerns using dedicated hardware are more pronounced.

The issue of representation will receive more attention in Chapter 6, when industrial exposure of the method will examine those representations of most relevance to the stakeholder-oriented assessment of interoperability and extensibility.

The guidelines provided for stakeholder identification and representation provide a concise answer to the first two research questions. Following some requirements for the assessment method itself are presented based on the problem analysis so far.

4.7 General assessment method requirements

Based on the stakeholder, and representation issues listed above, and the practical, industry-oriented nature of the research – the following set of general requirements are established for the proposed assessment method:

1. The goal of the method is to enable product-family stakeholders assess the architecture to ensure that it fulfils their requirements for the system qualities (see research aim, and framework)
2. The method shall be used by product-family stakeholders and especially those involved with the strategic business and technology aspects of the family
3. The method shall assist the stakeholders to carry out early life-cycle, requirements-architecture reviews so that the requirements and architecture become explicit and debatable for the family development team as early as possible.
4. The method shall build-on the SEI-backed SAAM approach towards architecture analysis (see section 4.4.1), with the following important contributions beyond the original SAAM method:
   - 4.1. Addressing **strategic family stakeholders** and their requirements with respect to **family-relevant** system qualities
   - 4.2. Providing **practical** advice and techniques on how the **industrial participants** (themselves) can provide the artefacts (deliverables) necessary for the assessment process.

[8] SOURCE: private conversation with Prof. D.K. Hammer October ‘00

4.3. Concentrating on the **direct involvement of stakeholders** in the assessment – particularly during requirements identification and generation.

5. For operationalisation – the method shall concentrate on the system qualities of interoperability and extensibility.
6. The method shall provide advice and techniques to help stakeholders communicate about requirements and architecture.
7. It shall be closely aligned to state-of-the-art family, architecture, and development-process management (this aids acceptance and implementation).
8. It shall provide accessible techniques for different stakeholders so that they can produce the required deliverables.
9. **Constraint** – The SAAM method [Kazman, *et al*., 1996], and the 4+1 views of Kruchten [1995] will be used as the basis of the method process and architecture representation, respectively.

4.8 Summary

This research focuses on assisting those stakeholders responsible for the strategic aspects (system qualities) of information-system families interact with the architecture. This support is in the form of an assessment method that is intended for early-phase architectural development. The method is requirements and use-case driven; which reflects the essential character of product-family architecting, and is aligned with emerging software development, and architecture-assessment, practice. The method is based on the SEI-backed SAAM method, which is emerging as a *de facto* assessment standard – and aims to extend this in order to:

- address family stakeholders, and family-relevant system qualities;
- provide how-to guidelines and techniques to enable stakeholders to provide the necessary assessment artefacts (requirements, architecture representations);
- provide quantitative metrics or checklists to identify and generate requirements;
- increase the involvement of the stakeholders with the architecture (assessment).

The assessment method is intended to be incorporated in the normal development cycle; and therefore uses existing practice (processes, representations) as much as possible. The method is operationalised here for the important qualities of interoperability and extensibility, which address the variety and temporal aspects of families.

Interoperability is defined as:
The ability *of two or more systems or components to exchange information and to use the information that has been exchanged in order to support a set of coherent business processes which require the co-operation of the systems or components*.

Extensibility is defined as:
The ability of the system to accommodate the addition of ***new*** *functionality*.

In order to identify the users of the method, and their most important content and representation requirements – the first research question presented in Chapter 2 has been addressed. These facts have been combined with the conclusions above and comprise a set of requirements that provides the needs and constraints the assessment method must satisfy. The next chapter describes the design of the Family Architecture Assessment Method (FAAM) to fulfil these requirements.

Chapter 5

Design – FAAM process and techniques

"Beware of sophisticated techniques; better work is done with simple tools in skilled hands" – Plossl and Welch, 1979, pp 13

5.1 Purpose

The purpose of this chapter is to present the overall design and accompanying rationale of the outline assessment method developed in response to the context, requirements and constraints on architecture assessment indicated in Chapter 3. This will include using (and extending) the answers to the first research question as presented in Chapter 4, to address the remaining research questions raised in Chapter 2, i.e.:

2. **How can the architecture be assessed with respect to the system-quality requirements?**
3. **How can the results of the assessment be recorded and communicated back to the stakeholders?**

The assessment method will be described including the concepts developed to aid stakeholder involvement and requirements specification, and the process steps to be followed. Implementation of the method via industrial case studies and the accompanying feedback for refinement will be described in Chapter 6.

5.2 Structure

The chapter is organised as follows: Initially the design principles underlying the method will be described. These principles are motivated by the requirements analysis, and derived from applicable product family, quality, software development theory. Hereafter the high-level design of the assessment method is presented, with extensive reference to the research questions, and traceability to the requirements. An example assessment case study is then presented to illustrate the method steps (more robust examples to evaluate the method are provided in Chapter 6). The chapter concludes with a summary.

5.3 Design principles

The principles underlying the assessment design are described in this section. Firstly the design goal is reviewed.

5.3.1 Design goal

The goal of this (development) phase of the research is:

to develop an outline architecture assessment method (guidelines, checklists, and process)

enabling information-system family stakeholders to assess how well the family architecture addresses their requirements with respect to strategically important system qualities. The initial method design presented here is operationalised for the system qualities of interoperability and extensibility.

While the method is optimised to treat the mentioned qualities, and specific techniques are provided for this purpose; it is postulated that the overall process (because of its basis in SAAM) and some techniques (those to highlight family issues) should also serve as a platform to address other qualities. Consideration of such issues is not addressed further in the method design and no claims to transferability to other domains can be made at this stage. This topic is revisited in the research evaluation in Chapter 7.

5.3.2 Design Principles (DPs)

Design principles are derived from the method requirements presented in section 4.7, and represent design-oriented aspects of the problem used to drive method development.

5.3.2.1 DP1 – Support for strategic family stakeholders

The method is intended for those strategic stakeholders in information-system family development indicated in Table 4.5. These stakeholders have traditionally been ignored by conventional software development methods; they are therefore not familiar with close interaction with development staff[1] in relation to (strategic) system qualities. These stakeholders need guidelines to help them understand the system qualities and to identify the change-cases that exercise the qualities (i.e. interoperability and extensibility). These guidelines should be in application or commercial terminology to be of most use to the stakeholders.

5.3.2.2 DP2 – Apply structured methods to requirements identification/generation

The assessment method recognises the strength of repeatable, structured steps in any assessment process. While assessment has connotations of measurement against a repeatable, quantitative scale – such quantitative knowledge is rare in current practice at the architectural level [Abowd, *et al.*, 1997]. However, this method aims to contribute to current practice by using where possible structured (e.g. checklist-type advice) guidelines to aid stakeholders in identifying important change-cases.

5.3.2.3 DP3 – Pay attention to family-oriented requirements representation techniques

Representation of family-related aspects of system qualities is practically non-existent in mainstream development. This method will develop and reuse straightforward techniques to provide graphical support for family stakeholders in expressing requirements, e.g. migration-

[1] In fairness to the strategic stakeholders, most development staff are not overly familiar in dealing with system qualities anyway.

maps to depict the various evolutionary possibilities provided to customers of the family across its life cycle.

5.3.2.4 DP4 – Stakeholder–architect interaction is an integral part of family-architecture development

In contrast to some assessment approaches – this method believes that architect-stakeholder dialogue during the review is an important part of exploring the architecture (through the interaction) and also increasing the understanding and ownership of the architecture throughout the family-development team. The method can be regarded as *encouraging* requirements-based communication between stakeholders and architect.

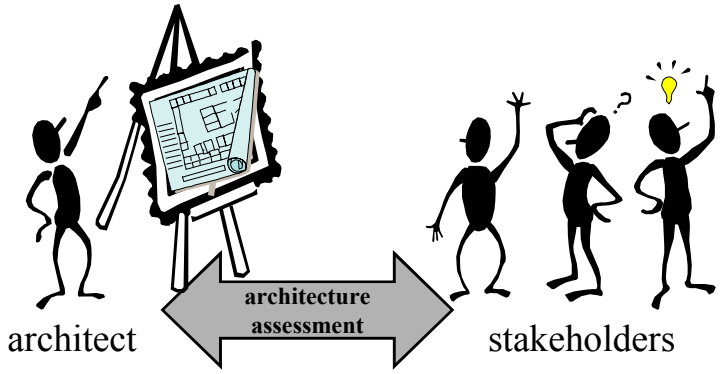


Figure 5.1: architecture assessment encourages architect-stakeholder dialogue

5.3.2.5 DP5 – Architecture rationale should be explicit for the family-stakeholders

The assessment method is positioned in the research framework as facilitating the clarification and communication of the rationale underlying the architectural decisions. Documented rationale is widely perceived as important [Abowd, *et al.*, 1997] in ensuring architectural quality by providing a reasoned explanation of why particular choices have been made. This is important when, in light of future modifications, many of these decisions will be questioned (i.e. during an assessment). Gacek, *et al.* [1995] have stressed the role of requirements-based rationale as proving an explicit link between the requirements-question and the architecture-answer. Capturing rationale after-the-fact is very difficult, especially when the people involved may have moved on.

5.3.2.6 DP6 – The assessment method will integrate with existing development practices in the producing organisation

As argued in Chapter 4, institutionalised assessment as part of the normal development process is the best means to increase architecture awareness and quality. In order to be unobtrusive to normal development, this assessment method will adopt the strategy to reuse as much existing (local-for-local) development practice and notation as possible. Where architecture-related practice is less established the method will:

- recommend use of emerging assessment wisdom (e.g. build on the assessment basis laid by the SAAM method);

- avail of development standards (e.g. an iterative development cycle as recommended by the Unified Process [Jacobson, *et al*., 1999], and UML [Rumbaugh, *et al*., 1999] for architecture representation).

5.3.2.7 DP7 – Industrial case-studies will be used to develop, test and enhance the method

This research design adopts the iterative-development principle from modern software engineering (see [Gilb, 1988]). Industrial case studies addressing extensibility and interoperability, with radiology and enterprise information systems, respectively, will be used to incrementally enhance the method to industrial strength.

5.3.2.8 DP8 – The method is aimed at the early-phase in family development

The usefulness of architecture assessment practice throughout the complete family life cycle is recognised, however, its role in early family and architecture definition receives most attention here. The method is meant to ensure that commercial-technical mismatches are identified as early as possible (at least before significant funds have been invested in implementation). The requirements and architecture must be sufficiently elaborated so that a reasonable discussion can occur; but the architecture, in particular, should not yet be institutionalised as the accepted development blueprint. The message is that there must be something to negotiate on; yet still room for negotiation (see Information Box 5-A for a more extensive discussion on this topic).

5.3.3 Assessment method – context

This section will provide the context within which the assessment method is framed. As depicted in Figure 5.2, the method is built on the fundamental work ongoing at the SEI on general architecture assessment. This research focuses on providing "how-to" advice to enable development-teams to conduct their own *self-assessments* as a means towards continuous improvement. It is recognised that useful advice regarding implementation of quality assessment is dependent on the application domain and the particular qualities involved. The approach here is to develop domain and quality "*add-ons*" (see middle-, and top-layer of Figure 5.2) which integrate with the general process and extend it to provide practical techniques[2] to fulfil the steps. The extensions described focus on:

- tailoring the general process for the domain of information-system families;
- adding practical techniques to support participants in generating the necessary process artefacts for the qualities of interoperability and extensibility in the domain.

[2] These techniques are based on questions, metrics, and representations which, after repeated exposure and refinement, will evolve into a good-practice process handbook; similar to those in use in more mature industries e.g. chemical, building

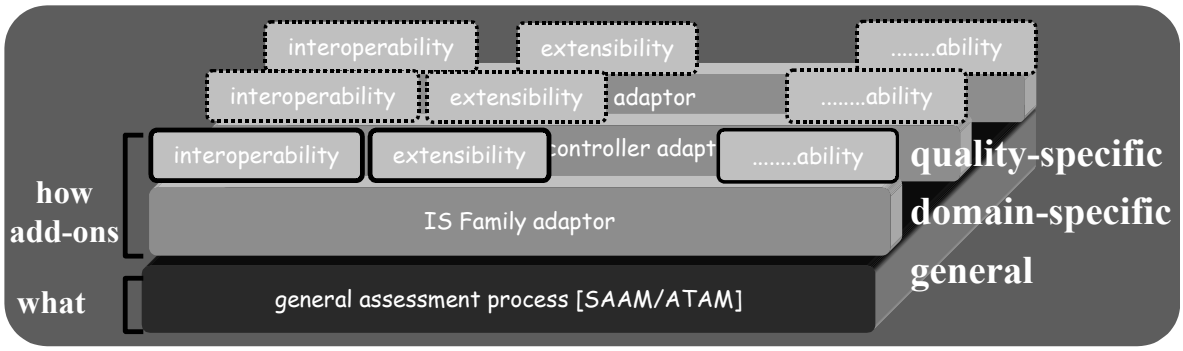


Figure 5.2: add-on based approach to architecture assessment

These are initial steps on the road to providing a comprehensive set of domain-, and quality-specific architecture assessment techniques. The long-term aim is that through increased industrial and academic exposure in these (interoperability and extensibility for IS families) and other domains and qualities, those domain-, and quality-specific parts will mature to add-on components to a general assessment method [Dolan, *et al*., 2000]. This depends heavily on adoption, and experience reporting, by the practising community.

As extensions are provided by others (dashed-add-ons in Figure 5.2) new possibilities for combining domain extensions may emerge, and as the general understanding of the system qualities increases, it may be possible to derive domain-independent techniques.

Information Box 5-A: A note on assessment timing – discovery or validation, …or both?

A very important aspect of architecture assessment is the *phase* in the family development cycle in which it occurs. Architecture assessment has been positioned as a cost-effective method with which to provide "*early*" feedback to stakeholders on the alignment of the family architecture with the business goals – this begs the questions: How early is "early"? and how does this impact the assessment process and results?

Best practice [Abowd, *et al*., 1997] recommends that assessment should begin as soon as requirements are *established* and there is a proposed architecture *in place* to review – however, there are no general criteria prescribed on what constitutes "*established*" requirements and when an architecture is "*in place*".

The same source reports that AT&T (no doubt in recognition of the iterative nature of requirements and architecture evolution) have also institutionalised a so-called "***architecture-discovery***" review – which is held after requirements have been *set* but before architectural decisions have become *firm*. This presents a chance for the development team to challenge requirements on the basis of their overall feasibility with respect to the general architecture directions proposed. The prime assumption of discovery reviews is that some preliminary architectural decisions have been made, but these are not firmly bound and therefore there is some leverage possible in guiding architectural development (and presumably in negotiating requirements). Discovery reviews are more requirements-focused than architecture-focused (simply because there is less architecture to focus on). They are therefore cheaper, but perhaps perceived as less beneficial because of the absence of a concrete architecture e.g.

they cannot provide definitive answers on performance issues because typically a binding to hardware does not yet exist.

Purists may quibble on whether this is a requirements review or an architecture review; the important point is that it is obviously useful to involve architects and stakeholders in the review of requirements as early as possible so that "real" requirements can be distinguished from "gold-plating", and requirements-architecture conflicts can be signalled as early as possible. Other important points to remember are:
this "**discovery**" review should be held in addition to (*not* in lieu of) the validation-oriented review advocated by the broad architecture assessment community (recall the performance example above);
discovery reviews are a necessary and normal part of new family *creation* where neither the requirements nor architecture is well established.

The conventional approach to architecture assessment is that of "***validating***" some concrete architectural decisions against strong quality requirements. In this case changing requirements or architectural decisions has serious consequences because the organisation has made a larger investment in both. Here the focus is more on the architecture and establishing its conformance to "given" requirements. With respect to the discovery review, two important differences can be identified:

- the validation reviews require the presence of a concrete, documented architecture and therefore occur *after* discovery reviews, and are a natural progression of architecture assessment through the life cycle. In fact a discovery review may be positioned as a pre-cursor to the validation assessment – used to pre-empt the inevitable requirements-centred discussions that are a natural part of the validation assessment.
- validation reviews are typical of more established families (and hence family-architectures) where the quality requirements are motivated by family *evolution* rather than family creation.

The position taken in this work is that an architecture will undergo different degrees of change during its useful life (see the simplified view of the architecture life-cycle in **Figure 5.3** below): from the large investment of initial creation, through the small-scale refinements and extensions of maturity (where changes are driven by periodic evolution), to the increasing efforts needed to maintain the architecture during degradation, finally the cost of "stretching" the architecture becomes so high that it is retired, and a new architecture is initiated. Assessments against requirements are useful for all proposed changes and architecture assessment is therefore used throughout the architecture life-cycle.

In conclusion: both types of assessment are useful, especially in the case of family creation where requirements uncertainty and the absence of concrete architectural decisions is typical of the early family life cycle. Discovery reviews naturally precede validation assessments and will become less frequent as the family matures. As regards assessment process: discovery reviews place most emphasis on requirements analysis and will produce more general guidelines aimed at *architecture creation*, whereas validation assessments concentrate on architecture analysis and provide more architecture-specific findings geared towards *architecture adaptation* while the architecture is maturing or starting to degrade.

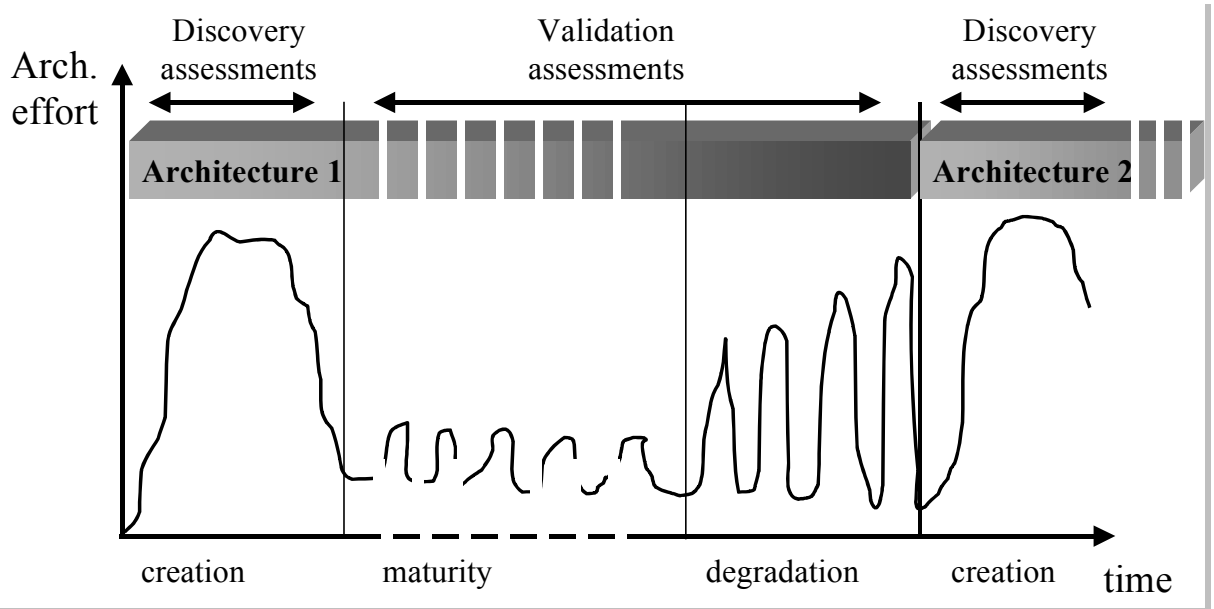


Figure 5.3:Discovery and validation assessments in the architecture life-cycle

5.4 Assessment method – overall design

This section presents the overall design of the assessment method (see Figure 5.4). The process ("what" aspect) will be described in the chronological order of process steps. Each step will be analysed for particular issues which must be addressed by those participants involved, and the characteristics of a practical technique ("how" aspect) to address these issues presented. Hereafter appropriate techniques are designed to implement the process step.

This approach separates the process (the steps and the issues) from the instruments (the techniques), recognising that the specific implementation of the techniques may be customised to individual organisational situations. Technique design (section 5.4.2) includes a description of the design knowledge used and decisions made.

This expected customisation of techniques reflects the general approach to FAAM-method development indicated in section 6 of Chapter 2. This is an outline method, which should be adapted and customised in response to general architecture-assessment developments and specific organisation experience. It should be regarded as a framework of practical guidelines and techniques, the rigour of adherence to which is at the discretion of the professionals using it (see [Van der Zwaan, 1998]).

The main part of this section (5.4.3) describes the method activities, participants' actions and technique application. Further details of the "how-to" techniques are provided in section 5.5 of the method design, and are the prime aspect of the assessment method that seeks to enhance current practice. The techniques and their usage in the process are synthesised from the previous process description, and illustrated using a real-life example assessment. The intention is that a high-level view of the method and techniques is described in the example

here, while a more extensive application including detailed technique-application and real-life feedback is provided in the following chapter.

5.4.1 Architecture assessment method – analysis

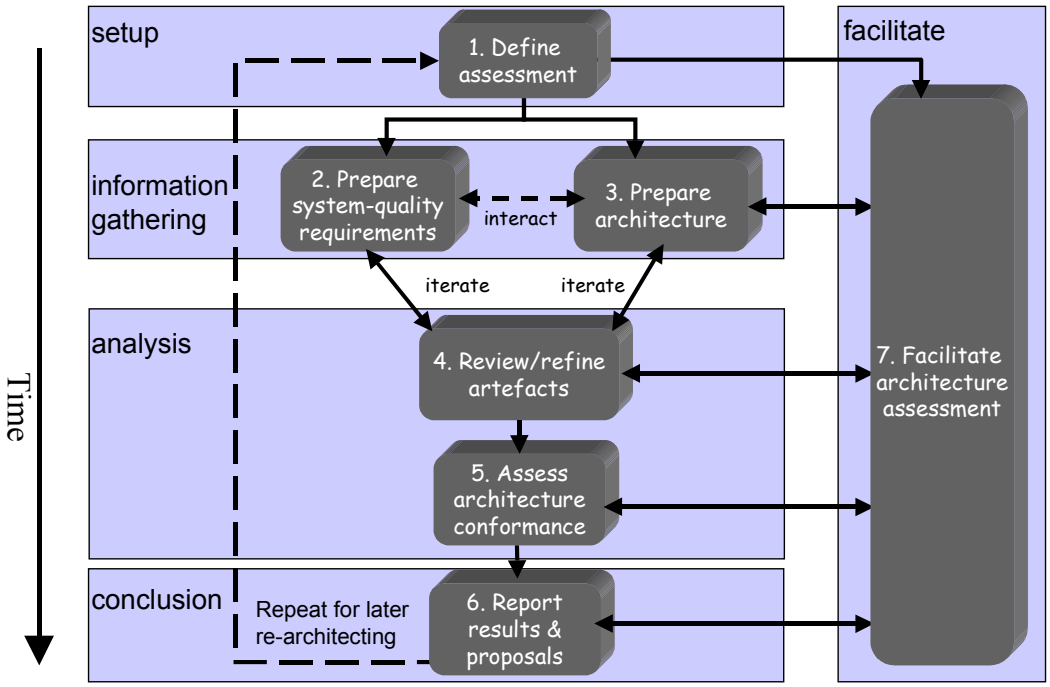


Figure 5.4: Family Architecture Assessment Method (FAAM) – process

Figure 5.4 above provides a high-level view of the steps to be followed in the proposed assessment method. The main process steps are motivated by SAAM [Kazman, *et al.*, 1996], the SEI report on architecture assessment best practice [Abowd, *et al.*, 1997], and ATAM [Kazman, *et al.*, 1998]. A particularly important aspect of this approach is the fact that the organisation participants (from the family development team) are responsible for preparing most of the artefacts used during the assessment, more details on this are provided later. The importance of ensuring that the development team members are focused on the *content*, motivates the role of facilitator to steward the *process*. This role of external (to the group) facilitator is well-accepted in assessment theory and practice (see e.g. [Kusters, *et al.*, 1999]), and plays an important part in:

- asking politically-naïve questions;
- handling the logistics and administration tasks;
- acting as a process expert to ensure the assessment progresses, thus enabling the other participants to concentrate on the product content;
- managing the human interaction to give each their say and steer clear of deadlock and negative conflict.

Clearly confining the facilitator to process guidance, and excluding him from preparing assessment artefacts is a marked deviation from reported architecture assessment practice (e.g. [Bass, *et al.*, 1998]) and a key discriminator of the method developed in this research.

The remainder of this section will provide an overview of the main deliverables and important issues associated with each step. This is used as input to develop techniques to support the participants in completing the activities.

1. Define assessment

This is a basic scoping exercise to determine the focus and intention of the assessment. Involved parties from the producing organisation are the sponsor and the primary stakeholders as identified in Chapter 4. An important issue, especially with organisation starting out with families, it may be that there is an inherent desire to "do families", but concrete, documented ideas as to what the family issues are may be missing [DeBaud and Schmid, 1999].

Challenge - 1. Establishing the scope and content of the family with the stakeholders

Besides the static-aspect of family content alluded to previously, significant emphasis has been made throughout this research report to the long-term, future-oriented nature of family thinking. This aspect is typically weak in those coming from the single-product approach, and very important for addressing the required interoperability and planned extensibility of the family as outlined in Chapter 4.

Challenge - 2. Establishing the future plans for the (interoperability and extensibility of the) family.

A potential deficiency in SAAM highlighted in both Chapters 2 and 4 was the need for some way of helping stakeholders to generate their requirements. This issue has been mentioned in Bass, *et al*. [1998] pp203, where the (external) assessors prepared their own scenarios in case stakeholders discussion needed to be stimulated. It is especially relevant when aiming to address the system qualities ignored by conventional software methods (see section 3.6.3.1 earlier), and to encourage stakeholders to frame their requirements in a (perhaps unfamiliar) family context.

Challenge - 3. Providing guidelines to stakeholders to help generate requirements.

Establishing the priority of competing requirements is necessary in order to concentrate on the most important issues when resources and time are limited. Most published literature (e.g. [Kazman, *et al.*, 1996]) emphasises the importance of prioritisation but is less clear on techniques to achieve consensus on this, especially among a varied group of family stakeholders.

Challenge - 4. Provide guidelines on prioritising competing requirements for assessment.

2. Prepare system-quality requirements

One of the most difficult challenges with respect to requirements – identifying and prioritising them – has already been mentioned. However, in order to actively encourage more stakeholder participation (see method requirements in the previous chapter), stakeholders need some structured means to record and communicate their requirements among each other and to the architect.

Challenge - 5. Provide a means for stakeholders to represent their requirements in a structured, assessment-ready manner.

3. Prepare architecture

This step is concerned with getting the required architecture representation available for explanation and assessment against the stakeholder requirements. The frequent lack of comprehensible (or even any) architecture documentation has been noted by many authors (e.g. [Bass, *et al*., 1998]).

Challenge - 6. Provide guidelines for architects in representing their architectures

4. Review/refine artefacts

The goal of this activity is to come to agreement on the requirements and architecture to be carried through to the subsequent assessment step. A challenge here is to establish the business, contextual or simply logistic constraints that may influence the operation of the assessment.

Challenge - 7. Clarify any other criteria or constraints influencing the assessment

5. Assess architecture conformance

This step is well addressed in SAAM, and indeed existing SAAM techniques can be reused here to help provide a structured record of the architectural insight gained.

6. Report results and proposals

Ensuring that the assessment results are accurately recorded, and more importantly communicated back to the organisation as part of the learning loop is the whole reason for the assessment exercise.

Challenge - 8. Provide a mechanism to record the assessment results and lessons learned

7. Facilitate architecture assessment

This is an ongoing activity throughout the assessment and depends on general assessment facilitation skills for success.

5.4.2 Technique design

This section contains an overview of the techniques designed in the method to address the previous issues. A detailed description of the techniques is provided later in section 5.5, and their usage in the overall method is indicated in section 5.4.3. In this section each of the issues introduced previously is revisited with an accompanying rationale showing the basis of the technique and explaining how it addresses the issue.

5.4.2.1 Issue 1 – family scope and content : TECHNIQUE – family-feature-map

Families are abstract, complex and not well understood in the information-systems domain. In order to establish what the stakeholders should be talking about some concrete representation of the family must be available. Those actively addressing software family research have adopted the tabular approach typical of the QFD [Hauser and Clausing, 1988] school. Variations on this theme are found in the Product Line Framework of Clements and Northrop [1999], the *product-maps* of DeBaud and Schmid [1999] and the *feature-graphs* of Bosch [2000]. Here an elementary variant of these approaches – the ***family-feature-map*** has been developed, providing an overview of the main functional and quality features of the family, and their allocation across the members. The choice for a simple overview is motivated by the facts that:

- the assessments are intended to provide coarse-grained insight into main risks (after [Kazman, *et al.*, 1996]) rather than detailed configuration-management information;
- the occurrence of the assessments early in the life-cycle means that the degree of knowledge is typically high-level.

Furthermore the variation possible is brought into consideration through identifying whether the features are core or optional per member. Incorporation of the notion of variety is motivated by the Generic Bill-Of-Material (GBOM) work of Hegge [1995] concerning the order-configuration and production logistics associated with product families. More details on the technique and its constraints are provided in section 5.5.1. For situations where the development organisation is more familiar with family concepts, and especially as repeated experience with architecture assessment matures, many of the other techniques referenced above may be adopted as supplements to the basic FAAM process.

5.4.2.2 Issue 2 – establishing the future plans for the (interoperability and extensibility of the) family: TECHNIQUES – family-context-diagram and migration-map

As noted here (see Chapter 3) and elsewhere, conventional software development methods have been weak in addressing system qualities, with the result that stakeholders are not practised or even comfortable discussion these issues. In order to support the stakeholders in formulating their requirements with respect to the system qualities of interoperability and extensibility, some simple visual representations of the issues associated with these qualities are proposed. These will have the role of clarifying and communicating the current situation to all stakeholders and stimulating a cohesive discussion regarding future changes.

The techniques provided by FAAM are the ***family-context-diagram*** and the ***migration-map***. The ***family-context-diagram*** is based on the context model of Structured Analysis [DeMarco, 1979]. Capturing relevant information from the problem domain or the environment of the proposed system-family members is well-accepted practice in all modern systems development. In Structured Analysis it has a formal notation and concentrates on capturing data flows between the system and *connected* external parties. In this method we follow (some of) the advice of Jackson [1995] and use the context diagram more generally to capture the elements of the problem domain, whether they interact directly with the systems comprising the family or not. Similarly, the connections between elements will not represent data flows but any general communication. This approach provides the stakeholders a view of the main elements in the system-family problem domain, both those that are connected to the family members and those that are not. These groups represent possibilities (i.e. potential change-cases) for *interoperability* and associated system-family *extension*. More details on this technique and its constraints are provided in section 5.5.2.

Lassing, *et al*. [2000] have also recognised the importance of context for information systems, and have promoted the idea the context diagram as an additional architecture view in its own right (the macro-architecture). They recognise two views: a *context view* providing a formalised treatment of the communications relationship with external systems; and a *technical infrastructure view* capturing the dependencies of the system on (shared) existing infrastructure elements. These views are regarded as a technical refinement of the ***family-context-diagram*** provided here and may be adopted as an alternative where stakeholders are comfortable with the notation. This is most likely as assessment practice in the organisation matures and where the architecture and requirements are more established (validation assessments).

The ***migration-map*** addresses the aspect of time, indicated early in this work as an important facet of product families. This technique is motivated by the family tree of human genealogy and provides a simple overview of the various members of the family and how they can be derived, or more-appropriately how they evolve, from one another. The importance to customers of being able to extend systems to follow business changes and to make future-proof investments has been discussed earlier in Chapters 3 and 4. The ***migration-map*** concentrates on intra-family extensibility (see section 5.3.2 in Chapter 4), examining the family members identified in the ***family-feature-map*** and supports the stakeholders in representing their planned evolution over time. This allows the family producer to prepare a roadmap the various family members, deciding which upgrade paths are possible for customers within the family. More details on the technique are available in section 5.5.3.

5.4.2.3 Issue 3 – providing guidelines to stakeholders to help generate requirements: TECHNIQUE – change-case-guidelines

In the FAAM method, as in general business operations, the family stakeholders are responsible for defining and managing the family requirements. The strategic, family-relevant concerns of stakeholders have been ignored in most conventional development methods and practices, and stakeholders are typically unfamiliar with their treatment. In order to help the stakeholders in focusing their requirements to the system qualities appropriate to the assessment (in this case interoperability and extensibility); and to help stakeholders generating the change-cases used to illustrate those requirements, simple guidelines are provided. The guidelines take the form of a structured questionnaire, which raises issues associated with a particular system quality. These guidelines are closely related to the earlier techniques, and provide textual support towards creating and utilising the graphical results of the family-feature-map, migration-map and family-context-diagram. An implementation of these guidelines for interoperability has been developed in the FAAM method, more details of which are provided in section 5.5.5.

The importance of supporting stakeholders in generating system-quality requirements [Dolan, *et al.*, 2000] has been recognised by other authors:

- Kazman, *et al*., [2000] describe a 3-D *scenario-elicitation matrix* to organise requirements (scenarios) based on stakeholders and system qualities.
- Bosch [2000] describes the idea of a *profile* (set of illustrative scenarios) per system quality, which is divided into *categories* (smaller subgroups) of scenarios as a means to motivate representative coverage. The categories are suggested by the architect's and assessor's experience of the particular system quality.
- Lassing, *et al*., [1999] focus on the risk-based identification of scenarios for modifiability assessment by allocating scenarios in a two-dimensional matrix. This is based on the likely *sources* of change (functional requirements, quality requirements, external components, external infrastructure); and the types of system *adaptations* (with external effects; from external effects, affecting relationships with other systems, affecting internal system structure, affecting shared components).

These are complementary to the guidelines for change-case generation herein; in fact the guidelines (for interoperability) will help stakeholders to generate change-cases for those external aspects of the Lassing matrix; or help in identifying categories for Bosch's profiles.

5.4.2.4 Issue 4 – provide guidelines on prioritising competing requirements for assessment: TECHNIQUE – requirements-ranking-criteria

As with any task involving multiple activities competing for scarce resources, the proposed requirements must be prioritised by the stakeholder group so that the most important issues receive the most attention. This will involve ranking (or even shedding) some requirements and these requirements will "belong" to some stakeholders. In order to get the stakeholder group to achieve consensus on setting priorities, FAAM provides ranking criteria that can be

used to provide an "objective", person-independent, means to order the requirements. Details and examples of initial criteria are provided in section 5.5.6.

5.4.2.5 Issue 5 – provide a means for stakeholders to represent their requirements in a structured, assessment-ready manner: TECHNIQUE – change-case-template

Conventional software development practice is dominated by attention to functional requirements; consequently associated specification techniques and stakeholder experience are mature. Stakeholders have not, however, the same degree of proficiency when it comes to specifying system-quality requirements, particularly in a product-family context. The provision of templates for specification is as old as (software) engineering itself and Gilb [1988], most notably, has advanced the application of templates to system qualities. The underlying principle of the FAAM method is that stakeholders themselves should generate and specify their requirements and evaluate the architecture accordingly. For this reason the advances in stakeholder-oriented requirements specification from modern (functionality-driven) software development – use-cases [Jacobson, *et al*. 1992] – are leveraged in the method through the similar *change*-cases; describing how the family should accommodate the changes underlying the system qualities. The ***change-case-template*** is organised to extract stakeholder knowledge relating to the qualities under review (here family interoperability and extensibility); and provides a standard manner to represent and communicate the requirements. Details of the use of the template and an implementation are provided in section 5.5.4.

5.4.2.6 Issue 6 – provide guidelines for architects in representing their architectures: TECHNIQUE – 4+1 architectural views

The architect needs a representation to communicate his architecture, both to the strategic family stakeholders central to the FAAM assessment and the larger set of developers and maintainers who will actively use the architecture in their operational tasks during family-system development. As indicated in 4.6.2.2, the intention of the FAAM assessment is communication; and any views and notations effectively supporting stakeholder-architect communication with respect to the qualities and systems under review is sufficient (see also [Bass, *et al*., 1998] Chapter 10). For this reason no architectural views or specific notation is mandated. However, where guidance is sought or where existing representations are deemed insufficient the ***4+1*** [Kruchten, 1995] approach addresses the most common aspects important for information-system architectures. Regarding the notation to use for the various views; again, the emerging UML [Rumbaugh, *et al*., 1999] standard is suggested, but other notations are possible, and based on the case-study experience reported later in Chapter 6, probable. Ongoing work in Architecture Description Language (ADL) research and increased insight into the treatment of system qualities, may provide more dedicated representation alternatives in the future.

5.4.2.7 Issue 7 – clarify any other criteria or constraints influencing the assessment: TECHNIQUE – assessment criteria

The prime criterion governing the architecture assessment is the stakeholder-generated change-cases discussed earlier. However there may be other important issues or constraints influencing the family outside the details of the system qualities under investigation. Examples are directions with respect to partnering, entering new markets, agreed technical policy or architectural constraints agreed with management. Even with respect to the change-cases, it may be possible, and useful, to note the bounds of negotiation for select change-cases. Explicitly capturing and using these extra criteria can help to focus resources and clarify discussion during the evaluation step of the assessment. Details and examples of such criteria are provided in section 5.5.7.

5.4.2.8 Issue 8 – provide a mechanism to record the assessment results and lessons learned: TECHNIQUE – conformance-statement

The main reason to invest the type of incremental, do-it-yourself architecture assessment advocated in this work is if the organisation producing the family obtains useful knowledge about the goals of the family, how the architecture supports these goals and what (if any) corrective actions are necessary to align the architecture and the goals. The ***conformance-statement*** is a mechanism to record the aims, constraints, findings and recommendations of the assessment. It is presented on behalf of the assessment participants to the assessment sponsor; and is used as input to make decisions or initiate re-architecting or re-definition activities for the family. The idea of a conformance statement is borrowed from the standards industry where it used to communicate the adherence of a product to a specific communication standard e.g. the DICOM [DICOM, 2000] image transfer standard in the medical imaging industry.

As regards other techniques, the SAAM metrics used to record and visualise:

- the number of architectural elements affected per change-case – ***Req.-arch. Interaction-metric***;
- (its inverse) the number of change-cases dependent on a particular architectural element – ***sensitivity-metric***.

are reused in FAAM.

5.4.3 Family Architecture Assessment Method (FAAM) – description

This section describes the FAAM method. The description provides details on the main process steps indicated earlier in Figure 5.4; especially regarding the sub-activities needed, the participants involved, the techniques used, and the artefacts produced. A graphical overview of each process step is included, and the legend in Figure 5.5 explains the symbols used. Practical advice with respect to using the techniques and general implementation guidelines are provided in the implementation guide section included after individual activities. As the major contribution of the method to the state of the art is in the requirements

definition phase, these activities receive most elaboration. This method description is the primary research output in terms of providing a general "how-to" oriented method for self-assessment of information-system family architecture by family stakeholders.

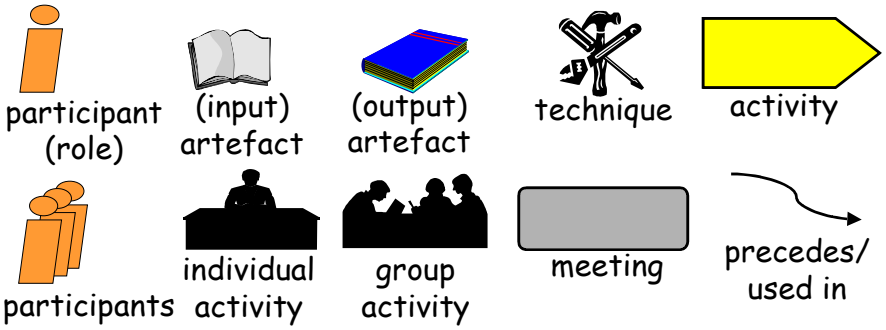


Figure 5.5: FAAM process legend

1 Define Assessment

Goal: to define the agreed scope and intention of the assessment with the family sponsor, architect and stakeholders, and to establish initial insight into the requirements and architectural artefacts.

Who: [*group/individual*] The facilitator (the sponsor, architect, stakeholders)

Why: This is necessary to ensure that all participants (especially the stakeholders) know what is expected of them and what they can expect from the assessment.

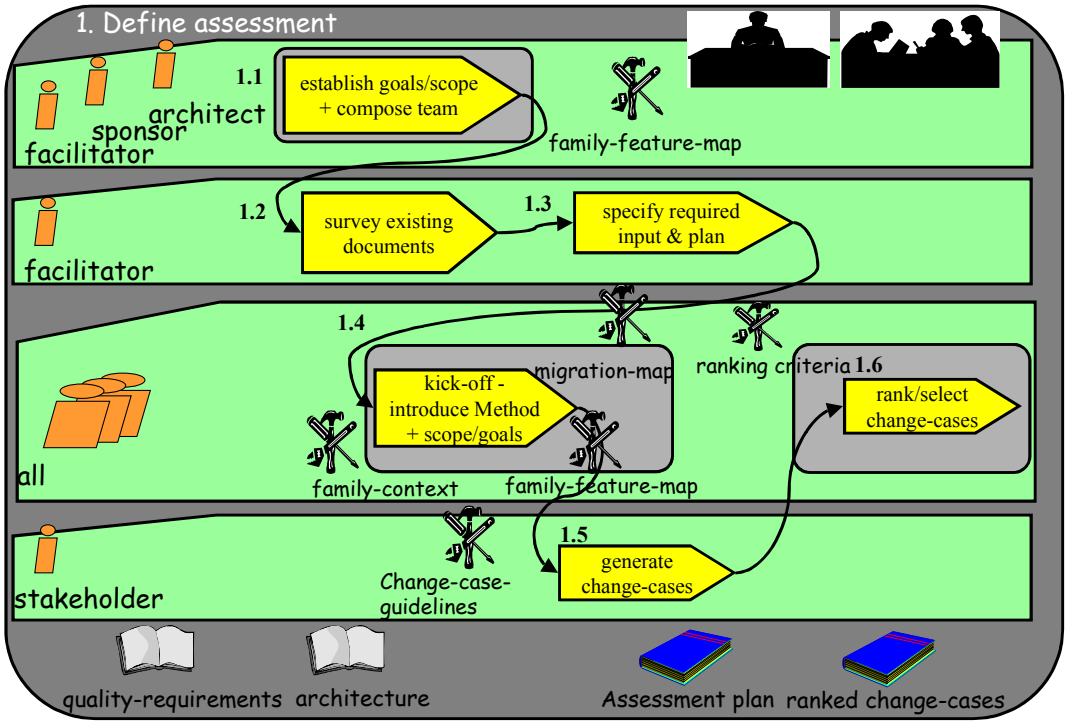


Figure 5.6: Define assessment: activity details

What:

1.1 Establish the goal, constraints and scope of the assessment with the assessment sponsor and architect, this must be communicated later to the remaining assessment participants from the development team (stakeholders).

- Establish the focus of the assessment on the system qualities – in this case interoperability and/or extensibility.
- Establish whether the assessment focus is requirements discussion "architecture discovery" review; or a formal architecture validation against a fixed set of requirements (see Information Box 5-A, previously). This will indicate where the focus of the assessment will lie, and is largely determined by the level and stability of the requirements and architecture.
- Establish the purpose and audience of the assessment report and what will be done with it. It is important to get management commitment and participant understanding that the evaluation is improvement oriented and not the basis of stakeholder or architect performance appraisal.
- Identify the assessment team composition:

- the four primary family stakeholders identified in Chapter 4 (business-management, product-management, customer support management, and development management) or their representatives;
- the architect (or architecture team);
- the assessment facilitator (to facilitate the assessment);
- application domain expert (to provide detailed application knowledge);
- external architecture expertise (optional, usual for a more formal evaluation);
- administrative and logistical support (optional).

1.2 Collect and survey available requirements and architecture descriptions to determine status and decide how long it will take to be ready.

- Existing documents related to the commercial, functional, quality and architectural aspects of the family should be reviewed to establish the level of information available on the assessment topics.
- It is particularly important at this time to establish whether there is enough information present in the organisation to proceed with the assessment, and if not to agree with the sponsor and the participants that this will be provided. The family-related techniques indicated earlier (***family-feature-map, migration-map, family-context diagram***) and the architecture are necessary for optimal assessment. If these are not available (not unusual where family experience is low) then they should be prepared before or in-parallel with the requirements and architecture specification activities.

1.3 Specify the required material (documentation, stakeholders) and initial plan for its presence prior to the assessment-session; this includes:

- Establish that the assessment participants are willing to take part, establish their roles, interests and associated assessment artefacts and techniques.
- Identify the required techniques and artefacts that may be missing and plan (with the participants) for their availability and the associated meeting schedule;
- Ensure the architect provides a description of the architecture – indicating computation and data components, and all component relationships (connectors) [SAAM-method step 1]. Typically for family-related issues the *logical* and *development* views from the 4+1 model [Kruchten, 1995] are important starting points in communicating the architecture. Software assessments are context sensitive [Punter, 2001], and precise details on appropriate views and granularity depend on individual situations.
- The plan (timing and resources) and decision to proceed should be agreed with the sponsor

1.4 Kick-off meeting

- In this plenary session the assessment goals and the method are introduced to the stakeholders. In particular it is important to relate the general goals of the family to the specific (interoperability or extensibility) focus of the assessment. It is important that the assessment sponsor plays a prominent part here.
- The overall method process should be introduced by the facilitator, with most emphasis on the remaining definition step and the requirements specification step. The plan for the remaining activities is also presented.
- The architect should present a general overview of the architecture to provide context to the stakeholders. This also provides the facilitator with an early idea of the architecture.
- The ***family-feature-map***, ***migration-map***, and ***system-context-diagram*** should be introduced here and stakeholders feedback obtained – this helps to build a shared view of the family and the main determinants of the system qualities.
- The stakeholders are informed that they will be requested to prepare one-line change-cases individually, based on the techniques provided and their own particular stake.

1.5 Generate change-cases

- The stakeholders individually generate change-cases (name only); based on their particular stake in the system. The ***change-case-guidelines, family-feature-map, family-context-diagram*** and ***migration-map*** (see section 5.5) are used here to identify candidate change-cases for interoperability and extensibility.

- As a rough guide, stakeholders should generate 5-7 change-cases as input to the following activity. The aim is not to curtail creativity or enthusiasm, but recognises that later pruning will occur and stakeholders must expect that some of their requirements will not proceed to the assessment session.

1.6 Rank and select change-cases

- The stakeholders meet as a group and introduce (in round-robin fashion) their short-form change-cases. The intention is to communicate the requirements to the whole assessment team. This is the first opportunity for the architect to hear the stakeholders requirements, and participants are allowed to seek clarification. It is important, however, to keep the focus – the aim is not to determine if the development project will meet all targets [Bass, *et al*. 1998], so extensive discussions regarding resource allocation, budgets and planning should be avoided.
- The task is now to reduce the (large, 5-7 change-cases per stakeholder) number of change-cases to a "top-10", for more detailed specification and subsequent architectural assessment. The need to reduce the number is for reasons of meeting effectiveness. Conventional assessment wisdom [Bass *et al*., 1998] pp 205, [Bosch, 2000] pp.109, suggests a maximum of 10 change-cases per assessment cycle, which is also close to Miller's [1956] magic-number 9 (7+2).
- Good advice for the *group* reduction process is described in Bass, *et al*., [1998] pp. 206, and adapted here:

- mark each candidate as likely/unlikely and remove unlikely ones;
- cluster remaining candidates into 5/6 groups (categories);
- rank each member of each group by criticality (see ***requirements-ranking-criteria*** section 5.5.6);
- (architect) ranks each member by its ability to reveal important aspects of the architecture;
- merge or reject candidates that do not significantly contribute to quality illustration or architecture understanding.

- Related (in the application domain or architecture) or conflicting change-cases are particularly important and their dependence should be reflected in the ranking and criteria setting.
- Selected change-cases are assigned to appropriate stakeholders for further specification.
- Again, the ***family-feature-map***, ***family-context-diagram*** and ***migration-map*** (see section 5.5) can be used to clarify discussion and help establish ranking in this activity. The facilitator has an important role here to balance time, stakeholder involvement and method progress.
- The ranking process depends heavily on the organisation culture (e.g. democratic, distributed or hierarchical, [Kazman, *et al*., 1999]) and can be laboured (see e.g. case-study experiences in Chapter 6). Supplementary techniques which may help

here include the overview provided by the requirements classification matrices discussed in section 5.4.2.3; or using a voting budget allocated to individual stakeholders as described in Clements [2000].

2 **Prepare system-quality requirements**

Goal: to specify the selected change-cases by the stakeholders, in line with the assessment goals.

Who: [*individual/group*] The stakeholders (the facilitator)

Why: Requirements are the primary input to the assessment process, and family system qualities have traditionally been unsupported by conventional methods. The stakeholders prepare details on their own requirements using the techniques and templates provided. The facilitator supports and guides stakeholders here.

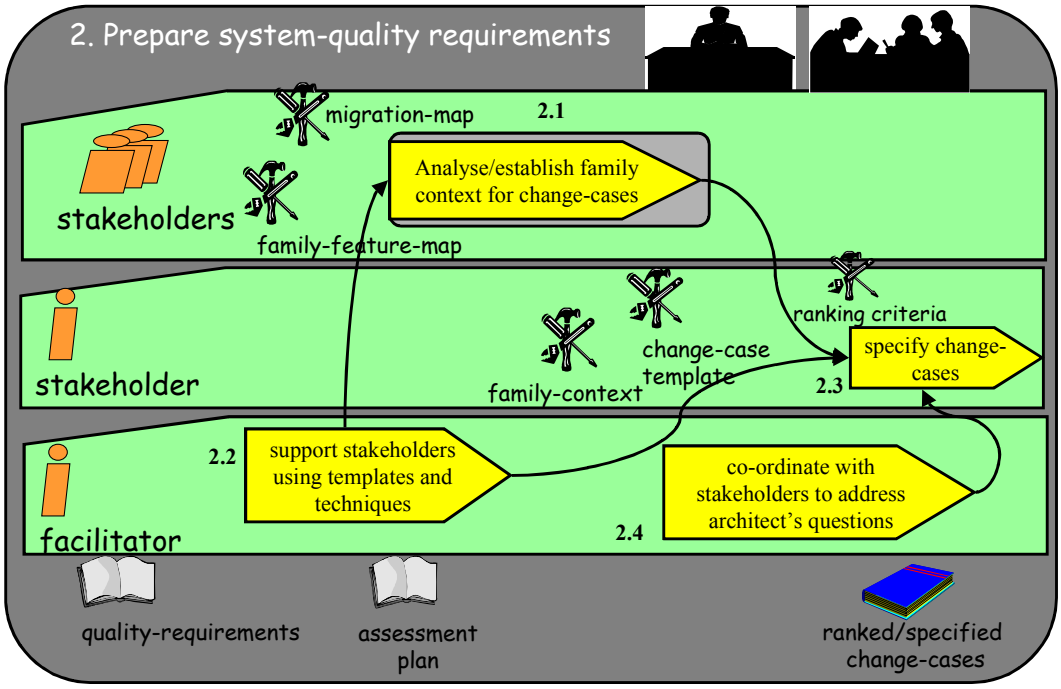


Figure 5.7: Prepare Requirements: activity details

What:

2.1 Establish the family context for the change-cases.

- As mentioned in the previous step, it is important to establish a definition of the family in order to provide context to the selected system qualities and their illustration via change-cases. The family-feature-map concept described in section 5.5.1; provides a useful initial overview of the scope and variability within the family. If *not already existing*, this should be prepared by (a selection of) the stakeholders and communicated to the complete assessment team to frame further requirements specification. This should be done *offline* from the main method process.

2.2 Support stakeholders using the tools and templates.

- The primary role of the facilitator is as a process and technique specialist. As organisations are acquiring maturity with architecture assessment the facilitator's support of participants in applying the FAAM techniques is very important.
- The facilitator will ensure that the ranking priority established in the previous definition step is reflected in the specification quality produced in this step.

2.3 Each individual stakeholder specifies the change-case(s) allocated in the previous step in more detail.

- First-hand specification of the requirements by the responsible stakeholders is the key feature of the FAAM method which helps transitioning the iterative, incremental application of architecture assessment to the family-development process.
- Fuller specification of change-cases for interoperability and extensibility is typically text-based and is structured for assessment use by a ***change-case-template*** as presented in section 5.5.4.
- The family-definition and quality-context techniques provided by the method: ***family-feature-map, migration-map; family-context-diagram***, and ***requirements-ranking-criteria*** can be used to help the stakeholder refine, or re-align their change-cases.

2.4 Co-ordinate with stakeholders to address architect's questions

- The facilitator liases between the stakeholders and architect during the parallel preparation of the requirements and architecture. Typically after the architect has seen the initial (short-form) change-cases in the previous step he will have some additional questions which this specification step should resolve. The facilitator acts to ensure that the specification provides enough requirement detail to address these concerns.

3 **Prepare Architecture**

Goal: to specify the architecture in line with the assessment goals, especially to identify the representations and rationale necessary to convey the architecture's conformance to the stakeholder requirements.

Who: [*individual*] The architect (the facilitator)

Why: The architecture is the assessment object and is visualised ("experienced") by the stakeholders through its representations. The architect must prepare the architecture to support the assessment based on the techniques and the initial requirements defined at step 1. The facilitator supports and guides the architect here.

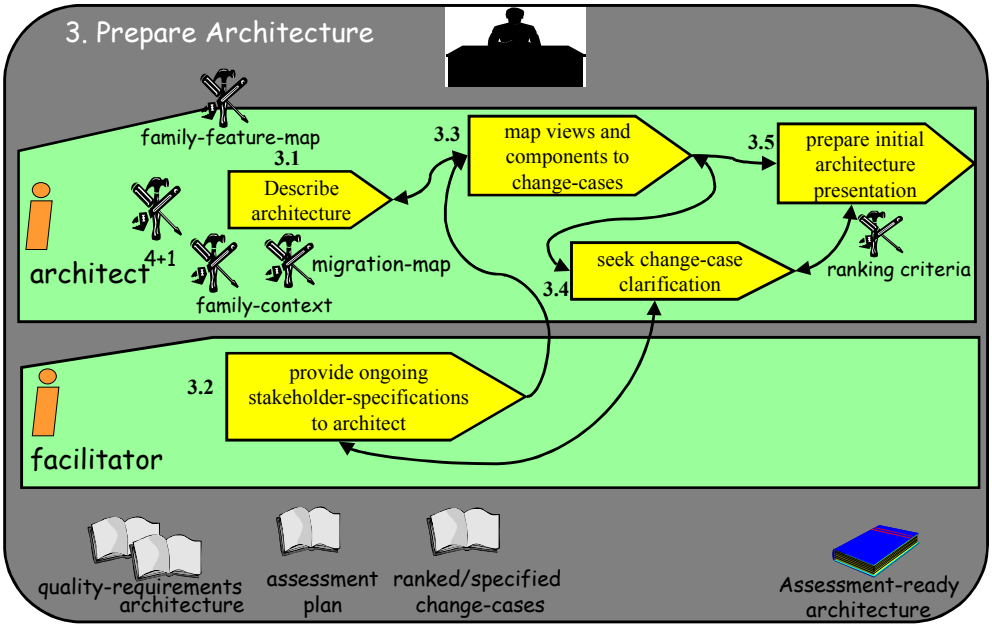


Figure 5.8: Prepare Architecture: activity details

What:

3.1 Describe the family architecture (if not already done) using existing representations in the organisation with an emphasis on the computational components, data components and their relationships.

- Existing views shall be embellished where needed with the information defined in Kruchten's [1995] ***4+1*** views (see Chapter 3 for an explanation of these views).
- Typically for system-quality issues in information-system families the following views have proven most useful in this research (see Chapter 6 for more details):
- logical view (information model) – static issues
- development view (software modules in development organisation) – static issues
- process view (e.g. UML Interaction diagram) – dynamic issues
- The family definition techniques (***family-feature-map***, ***family-context diagram***, ***migration-map***)and results mentioned earlier are used here to incorporate the family concepts into the architecture;

3.2 Provide ongoing stakeholder specifications to architect.

- The change-cases derived earlier guides the type of representations needed. These should be presented (even while still evolving from the stakeholders) to the architect so that a focused response can be prepared.
- This is an ongoing activity and will be driven by stakeholder specification and specific questions from the architect (see 3.4). It is one of the co-ordination steps provided by the facilitator between the requirements and architecture specification (see also step 2.4 previously)

3.3 The architect should record which changes-cases are related to which architecture views, and components.

- This indicates those views and components important for stakeholder concerns. This information is also used to calculate the SAAM-metrics (***req-arch interaction***, and ***sensitivity*** metrics) on the architecture.

3.4 Seek change-case clarification. The architect may react to the change-cases, to question priorities, or request extra details. This activity is related to co-ordination issues between requirements and architecture specification mentioned earlier.

3.5 Prepare initial architecture presentation

- The architect prepares for the assessment meeting by making a dedicated presentation showing which specific parts of the architecture support, or are affected by, the proposed stakeholder requirements.
- The requirements priorities as set by the ***requirements-ranking-criteria*** are used to manage the focus of the presentation.
- The precise views, their granularity, and completeness is the responsibility of the assessment-team – if the stakeholders are satisfied then it's sufficient, if not then it isn't. The approach in this method is that it enables *professionals* to carry out an assessment – the assessment-team is responsible for its actions. If the initial presentation is not adequate then this will be revealed in the assessment meeting (see step 5) and the information used to refine the description provided in the conformance statement, and as a general guideline for future assessments.

Steps 2 and 3 above are iterative and there is typically intensive interaction, but **not** architecture assessment between the stakeholders.

4 **Review/refine artefacts**

Goal: to review the material gathered in the specification phase, re-affirm the requirement priorities, resolve any open requirements issues and, if necessary, refine the specifications (previous step). The readiness of the team for the assessment meeting can be ascertained here and any necessary recovery actions initiated. At this point, sufficient clarity and commitment is established to develop a technical and organisational assessment-execution plan.

Who: [*group*] The facilitator (the stakeholders, architect)

Why: This is a very important step in the overall method – the requirements and architectural material must gathered, and the information shared. The participants must commit on which subset of requirements will be assessed, the clarity of the material, and any additional assessment criteria that will influence the outcome.

What:

4.1 Check architecture

- The facilitator and architect review the status of the architecture presentation, this is done to monitor progress – explanation of the architecture to the stakeholders is the topic of a separate assessment meeting (see next step).
- The facilitator and architect use their knowledge of the emerging change-case specifications and the ***4+1*** approach (Kruchten [1995]) to agree on the number and granularity of the architecture views used.
- Any unresolved questions from the architect are also gathered here to be put to the stakeholders.

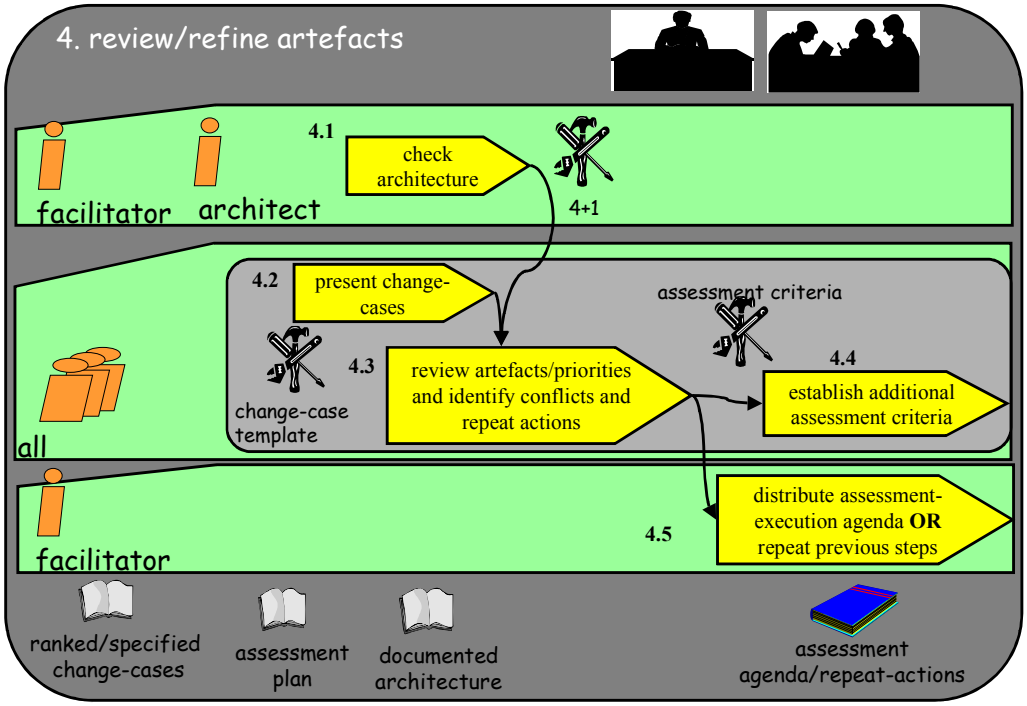


Figure 5.9: Review/Refine Artefacts: activity details

4.2 Present change-cases

- Change-cases are presented by the stakeholders to all assessment participants.

4.3 Review artefacts/priorities and identify conflicts and repeat actions

- Review material against technique-guidelines (e.g. ***change-case-template***) and participant opinion to ensure that the change-cases cover the requirements for each quality.
- Reaffirm the priorities, and check and resolve conflicts in the requirements. If important new requirements arise then they may be added and allocated to a stakeholder for specification – the goal is to address important requirements not

manage a list. The general family-definition and system-quality techniques may also be consulted to identify conflicts or manage scope.

- Any outstanding questions from the architect should be resolved prior to the upcoming assessment meeting.
- In case rework is needed, the appropriate participants may return to the previous specification steps and another review may be held before proceeding to assessment execution.

4.4 Establish any additional assessment criteria

- In addition to the change-cases some additional guidelines or constraints may be determinant in assessing the family architecture. These relate to organisational issues, business decisions, or architecture principles (see section 5.5.7 for some examples). It is important to get such "non-requirements" issues explicit so that the rationale is clear to all, before the assessment meeting.

4.5 Plan and distribute the assessment execution agenda or repeat previous steps

- Based on what is learned during the review: the plans (aims, material roles, venue, timing) for the assessment execution step are formulated and communicated. This can include any necessary rework, and a repeat review or one-on-one checks with participants may be needed.

5 **Assess architecture conformance**

Goal: to assess the conformance of the architecture to the quality requirements (change-cases) as specified by the stakeholders

Who: [*group*] The architect (the stakeholders, facilitator)

Why: Assessment of the architecture against the requirements is the aim of the exercise; and this step provides the architect with the opportunity to present the stakeholders with those parts of the architecture relevant to their requirements.

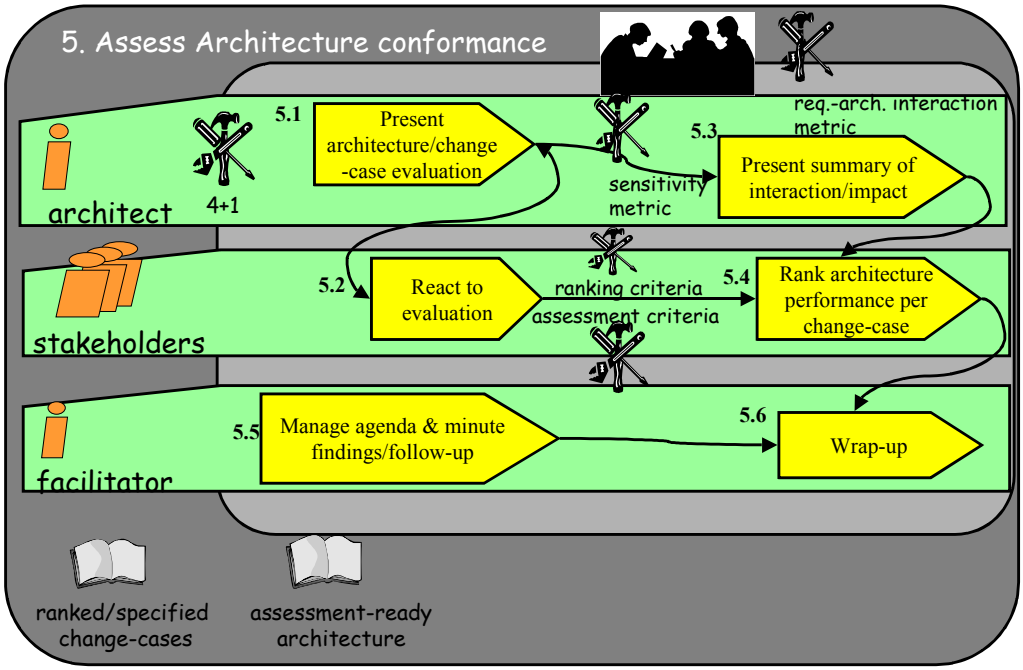


Figure 5.10: Assess Architecture Conformance: activity details

What:

5.1 Present architecture/change-case evaluations.

- The architect must explain using the (***4+1***) architecture representations how the architecture supports each change-case, and indicate whether the architecture supports the change-case without modification (direct, in SAAM terms), or whether modification is required (indirect, in SAAM terms). The explanation reveals the mapping between requirements entities and appropriate architecture entities; and is useful to help translation from the architecture to the requirements domain. This exercise also serves as a means to record the rationale for architectural decisions.
- Direct change-cases indicate that the architecture already supports the desired system quality – these change-cases are important in explaining the architecture to the stakeholders.
- Indirect change-cases must be analysed further for their impact on the architecture. The scale of the changes, the components to be changed or added, and their costs must be summarised for each indirect change-case [SAAM method step 3].

5.2 React to evaluation

- The stakeholders must interactively provide feedback to the architect. Typically this is to establish clarity on how the architecture addresses the change-case.
- The architect may also argue for technical reasons not to fully support some change-cases and the resulting architect-stakeholder interaction is precisely the aim of the exercise.

- Where changes are to be implemented (or not) it is important that the group understand the implications uncovered by the assessment.

5.3 Present summary of component interaction and architecture impact

- The architect closes his presentation with an overview of the aggregate impact of the change-cases on the architecture. This is best presented in a tabular overview listing change-cases, affected components and associated costs (including negative side effects).
- The ***req.-arch.-interaction-metric*** provides an overview per indirect *change-case* of those architectural components affected by the proposed change.
- The ***sensitivity-metric*** provides an overview per *component* of the indirect change-cases that affect it.
- Related components should be affected by related change-cases if the architecture has an appropriate separation of concerns. [SAAM method step 4].
- Components that are affected by unrelated change-cases (expressed in the ***sensitivity-metric***, and regarded as *trade-off points* in ATAM) *may* indicate a weak separation of concerns; or a very important component – again individual context is determining here.

5.4 Rank the architecture's performance per change-case.

- Weigh each indirect change-case and the change-case/component interaction in terms of their relative importance. This should be reached by group consensus. [SAAM method step 5]
- The architecture metrics and the pre-defined ***requirements-ranking*** and the general ***assessment-criteria*** are used to support this process, including creative dialogue towards compromise.

5.5 Manage the agenda and minute the discussion.

- The facilitator must ensure that the meeting flows and issues are dealt with in order of importance.
- The issues, metrics, important discoveries raised during the review should be recorded in minute-form by the facilitator (or appropriate administrative support) this will act as a log of the assessment and is will be input to the official conformance-statement.

5.6 Wrap-up. The facilitator closes the meeting by ensuring that the facts and findings are clear and that follow-up actions and unresolved issues are noted.

6 **Report results and proposals**

Goal: to document and report the assessment results to participants and other concerned parties; and to establish a baseline for follow-up assessment or architecture, requirements rework.

Who: [individual/*group*] The facilitator (the stakeholders, architect)

Why: This is the mechanism for reflecting on the assessment results, transforming the knowledge generated into concrete conclusions and (remedial) actions, and communicating these findings to the organisation.

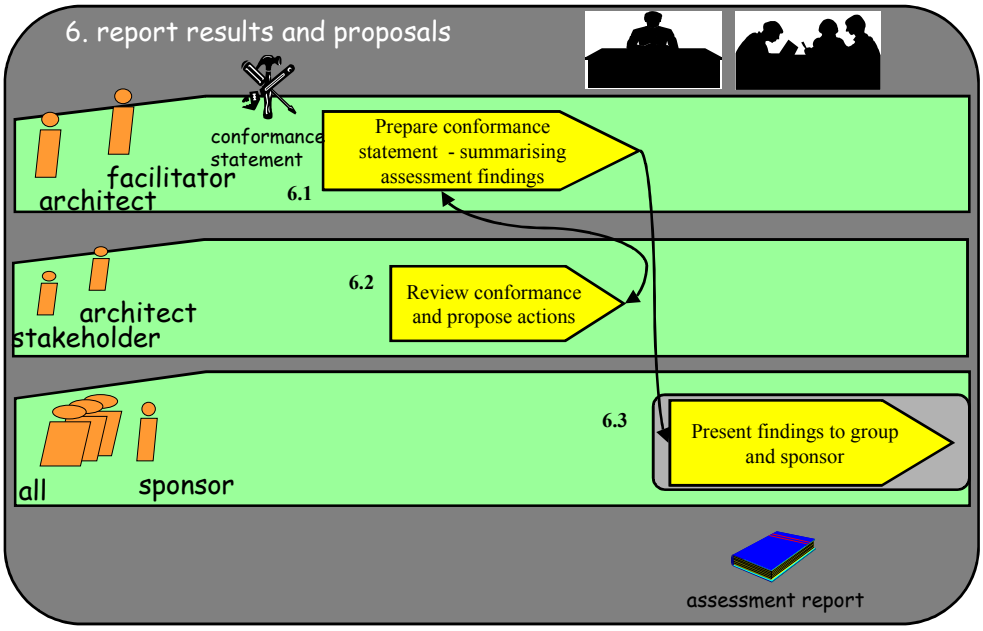


Figure 5.11: Reporting: activity details

What:

6.1 Prepare conformance statement.

- A summary of the architecture's accommodation of the required interoperability is provided by a ***conformance-statement*** (see section 5.5.11). This is prepared by the facilitator (with the close support of the architect) and is intended to communicate the assessment findings to the stakeholders and sponsors.
- The requirements-architecture mapping knowledge and metrics discovered during the assessment meeting will be important here in illustrating any system-quality limitations of the architecture.
- The weighted change-cases and interactions (see step 5.4) should be used as the basis of planning any *re-architecting* activities in order to improve architectural support for the indirect change-cases.

6.2 Review conformance and propose actions

- The considered findings of the architect and facilitator are communicated to individual stakeholders for discussion on follow-up actions.

- Re-architecting should only be considered in conjunction with stakeholder feedback. Important in analysing any changes is that the changes do not adversely affect any direct change-cases or other system functions/qualities – this is the architect's responsibility.

6.3 Present findings to stakeholder group and sponsor

- A set of proposals – including repeat assessments (see dashed-arrow in Figure 5.4) for modified or ignored requirements or architecture elements – should be presented to the stakeholders and assessment sponsor. These should be accompanied with rationale and risk-assessment for incorporation into ongoing family-planning discussions with stakeholders.
- This continued management of the stakeholder requirements throughout the (architecture) development process is the key principle underlying this incremental approach to architecture assessment.

7 **Facilitate architecture assessment**

Goal: To guide and support the participants in implementing and reporting on the assessment.

Who: [*individual*] The facilitator is an expert in the method and in general process facilitation, has a solid knowledge of architectural issues, but not necessarily in the application domain and – in the interests of independence – is external to the development team and the family.

Why: The assessment participants are experts in the application and technical domain of the family. In order for them to concentrate on these important issues during the assessment, the facilitator supports them in following the process, applying the techniques, and preparing the artefacts needed by the method.

What:

7.1 The general steps for facilitation have already been alluded to during method explanation, in general the facilitator works step-by-step through the method (see Figure 5.4) to support participants and to co-ordinate the communication and deliverables between steps.

5.4.4 Participant work overview

Table 5.1 below contains an overview of the activities the participants are involved in (based on the preceding method design), the estimated effort per activity-group, and the consolidated workload per stakeholder. This information is based on the experience aggregated across the case studies (see Chapter 6). These figures are *guidelines* to set expectations for initial adoption of the method (as the case studies were). Most effort is concentrated on the facilitator and architect, and the architect's workload may decrease as the architecture matures and the assessment experience increases. The expectation is that each stakeholder will provide 1.5 days to the assessment; again this is expected to reduce to 1 day as experience increases, and some of the group meetings can be shortened. The group activities are indicated by

shading and effort for the preparation of family-related techniques (e.g. family-feature-map) is not included.

Table 5.1: Overview of participants' activities and workloads

(S)takeholder **(A)**rchitect **(F)**acilitator **(Sp)**onser	**Type**	**Activity**	**Details**	**Effort [hrs.]**
Sp, F	1-on-1	1. Define assessment	1.1 establish scope and compose team	1.5
F	Private work	1. Define assessment	1.2, 1.3 survey documents, specify plan	8
A, S, F, Sp	Group session	1. Define assessment	1.4 kick-off	1.5
S	Private work	1. Define assessment	1.5 generate (5/7, 1-line) change cases	0.5
A, S, F	Group session	1. Define assessment	1.6 rank/select change-cases	2
S	Private work/ 1-on-1	2. Prepare requirements	2.3 specify change-cases (assume 2 change-cases per stakeholder)	2
F	1-on-1	2. Prepare requirements	2.2 support stakeholders 2.4 co-ordinate stakeholders-architect	8
F	1-on-1	3. Prepare architecture	3.2 provide requirements to architect	4
A	Private work/ 1-on-1	3. Prepare architecture	3.1, 3.3 - 3.5: describe architecture, map, clarify, prepare presentation.	12
F, A	1-on-1	4. Review/ Refine artefacts	4.1 check architecture	1
A, S, F	Group session	4. Review/ Refine artefacts	4.2 – 4.4 present change-cases, review artefacts, assessment criteria	2
A, S, F	Group session	5. Assess architecture	discuss/review architecture	3
A, (F)	Private work/ 1-on-1	6. Report results	6.1 prepare conformance report	4
A, S, F, Sp	Group session	6. Report results	6.3 present report	1
S	TOTAL			12 (**1.5 D**)
A	TOTAL			24 (**3 D**)
Sp	TOTAL			4.5 (**1/2 D**)
F	TOTAL			32 (**4 D**)

5.5 Assessment method techniques

This section describes the techniques discovered or developed during the research as practical aids in completing the architecture assessment method steps presented above. These

techniques represent the general contributions of the research towards extending SAAM and assisting stakeholders to provide the artefacts necessary for architecture assessment.

Table 5.2 below provides a summary of the various techniques used per method step.

Table 5.2: Overview of technique usage in method steps

Tools\Method-steps	1 Define	2 Req.	3 Arch	4 Review	5 Assess	6 Report	7 Facilitate
1. family-feature-map	✓	✓	✓	✓		✓	✓
2. family-context-diagram	✓	✓	✓	✓		✓	✓
3. change-case-template		✓		✓			✓
4. migration-map	✓	✓	✓	✓		✓	✓
5. change-case-guidelines	✓			✓			✓
6. requirements-ranking-criteria	✓	✓	✓	✓	✓	✓	✓
7. 4+1			✓	✓	✓	✓	✓
8. assessment-criteria				✓	✓	✓	✓
9. req-arch-interaction-metric					✓	✓	✓
10. sensitivity-metric					✓	✓	✓
11. conformance-statement						✓	✓

extensions to SAAM developed in this research

The techniques will be described in the remainder of this section, and illustrated using material from a small case study in the medical information system domain (a PACS system for image storage and distribution). The case study material is intended for demonstration purposes only and is derived from a small-scale implementation of the method: 3 participants, with two (interoperability) change-cases to assess how the family can address the requirements to:

1. Provide handling of ultrasound Images and measurement Data by PACS;
2. Allow connection of third party (DICOM) Workstations to the PACS.

In order to protect confidential information and to make the material accessible, simplification and alteration of details has occurred.

5.5.1 Family-feature-map

Definition

The ***family-feature-map*** is a tabular structure listing which features are contained in which family-members (variants). Further it indicates whether these features are a *core* part of the

variant (i.e. always present or enabled, and standard included in the variant); or whether they are *optional* (can be enabled depending on customer configuration).

Role

The ***family-feature-map*** is intended to provide a summary overview of the family. It highlights the commonality and variety between the various members of the family, and typically has a functional perspective. It provides stakeholders with a feature-based overview of the scope of the family, and provides architects with an indication of which features are "always" available and which are variable. This is a concrete definition of the family in application-terms, and provides participants the vocabulary with which to discuss the boundaries and internals of the family members.

With respect to the architecture assessments discussed here – there are two important points:

- practical experience shows that the exercise of defining the ***family-feature-map*** will *stimulate* the stakeholders to consider and generate change-cases.
- the absence of a clear ***family-feature-map*** makes it *difficult* to generate a useful family architecture – any weakness in the family definition is an early indication of weaknesses in the architecture.

Who

Typically the ***family business manager*** is responsible for defining the functional content of the family in response to functional and market requirements. Individual ***product managers*** (commercially responsible for individual variants) use the ***family-feature-map*** to co-ordinate their activities and limit their area of responsibilities.

Example

Functions	lite	Dept.	Ent.
Pat-reg	O	O	C
Request-reg	C	C	C
Appt.-sched.	C	C	C
Exam-admin	O	O	C
WLH	C	C	C
reporting	O	O	C
Mgt.-reporting	-	-	C
Review-prep	-	C	O
viewing	O	C	O
Archive-mgt-F	O	O	C
Archive-mgt-D	-	C	O

Figure 5.12: Sample family-feature-map showing core/optional function-elements in each of 3 family-members

Limitations

The family-map does not show the variability *within* a particular feature e.g. the different types of viewing functions supported (but this could be remedied by simply increasing the

amount of detail shown). It also does not show dependencies (either commercial or technical) between optional features. Such additional detail can prove necessary in the assessment – this is typically indicated by the architect in the iteration between the architecture and requirements analysis steps in the method.

5.5.2 Family-context-diagram

Definition

The ***family-context-diagram*** is a simple graphical-overview of the various systems/components which are (or can-be) present with the family in its operating environment.

Role

The ***family-context-diagram*** is used primarily as a means to:

- provide a real-life, product-based representation of the current family content and boundary in its application context
- a means to provide insight on the actual and candidate systems that *interoperate* with the family
- a means to indicate possible systems which could be incorporated/replaced by the family thus *extending* the family-scope.
- Act as a means to encourage stakeholder thinking on **interoperability** requirements by describing the candidate "actors"; and on **extensibility** opportunities by highlighting associated applications in the domain. This prompts stakeholders to generate both intra-, and extra-family interoperability and extensibility change-cases.

Who

Typically the ***product manager*** and/or the ***architect*** provide the context in which the family or individual products therein operate.

Example

The example in Figure 5.13 below illustrates the various internal (based on the suite-concept of family indicated in Chapter 3) and external components that provide the application context for the PACS (Picture Archiving and Communication System) image management system. The *ultrasound modality* highlighted (dashed border) as a variant of external system in the figure will be referred to later in this section when illustrating an example (extra-family) interoperability change-case.

Limitations

The ***family-context-diagram*** provides a static view of the systems (family members) in the application domain, it contains no information as to *how* interaction occurs dynamically, or specifies data/control interfaces and infrastructure dependencies. In early-design architecture assessments this is not a limiting factor for stakeholders. Architects will normally need such information to have a convincing story at assessment-time, and certainly this information is needed as system construction begins. As mentioned in section 5.4.2.2, the context-diagram is so important in information systems design that it has been elevated to a proper architectural view by Lassing, *et al*.[2000].

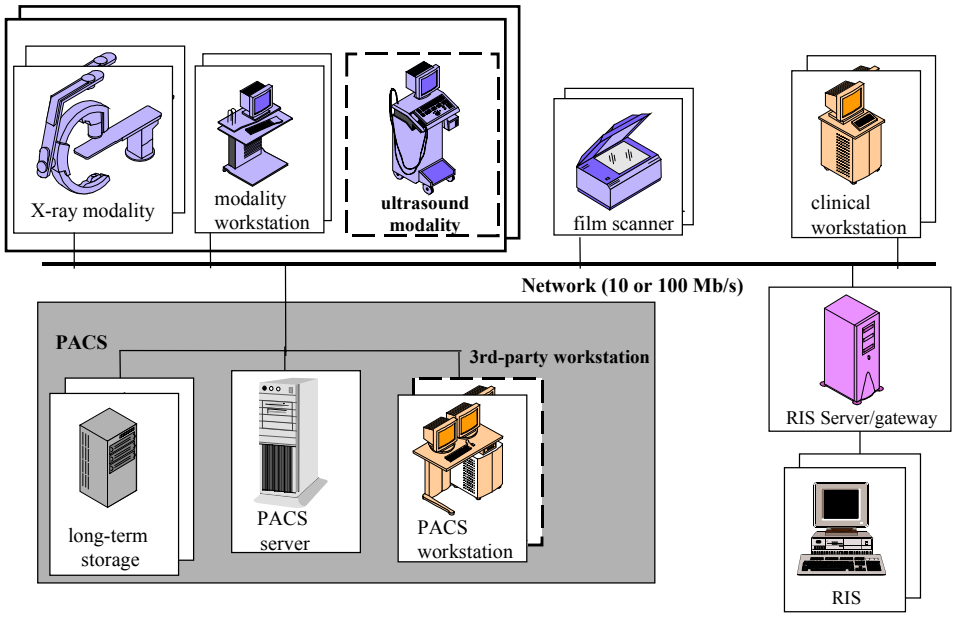


Figure 5.13: Sample family-context-diagram showing the internal and external components required to interoperate in the (PACS) family application domain

5.5.3 Migration-Map

Definition

The ***migration-map*** is a simple graphic illustrating the various members of the proposed family and their ancestors and planned-descendants over time. This tool captures the past/future aspects of family indicated as an issue in section 5.3.2.3 – analogous to a family tree in genealogy.

Role

The ***migration-map*** sets out the roadmap for the family – it is especially useful in encouraging business stakeholders to make the following important (especially for information system families) decisions regarding family continuity and **extensibility** explicit:

- What legacy systems must we migrate from and which (new) members are affected
- What level/type of migration is required above – examples
- What individual family-members are contained in the family and how/which members do we offer customers the ability to move/extend to within the family

Further it provides motivation for stakeholders and architects to pre-empt the risks and issues associated with legacy migration in the family – a very major issue in information systems community.

Who

Typically the ***family business manager***, ***customer support manager*** are responsible for defining the bridge between past and future to be supported by the family. The ***development manager***/***architect*** also plays a leading role here because of the need to account for the costs of technical discontinuities associated with trying to please all the existing customers and all the future customers. This is typically an area where commercial wishes and technical limits

conflict; it is, therefore, vital that the discussion is clarified early in the family's life thus increasing the chances of business and technical alignment.

Example

The example below is based on a *fictitious* product family (RIIMS) which integrates previously separate information-, and image-management systems. Without going into details; the main point is that the stakeholders have indicated which members belong in which market segments (entry, standard, enterprise), which legacy migrations must be supported, and therefore which not (e.g. *legacy-RIS* to *RIIMS* but not to *Basic RIIMS*). The diagram also captures the strategy that dictates which upgrades or extensions *within* the proposed family the producer will support. In many cases this is strongly motivated by the technical differences foreseen between entry-level and enterprise systems, and provides a means to optimise revenue streams by forcing customers to follow a planned path of expenditure, and reducing the technical gap between extensions.

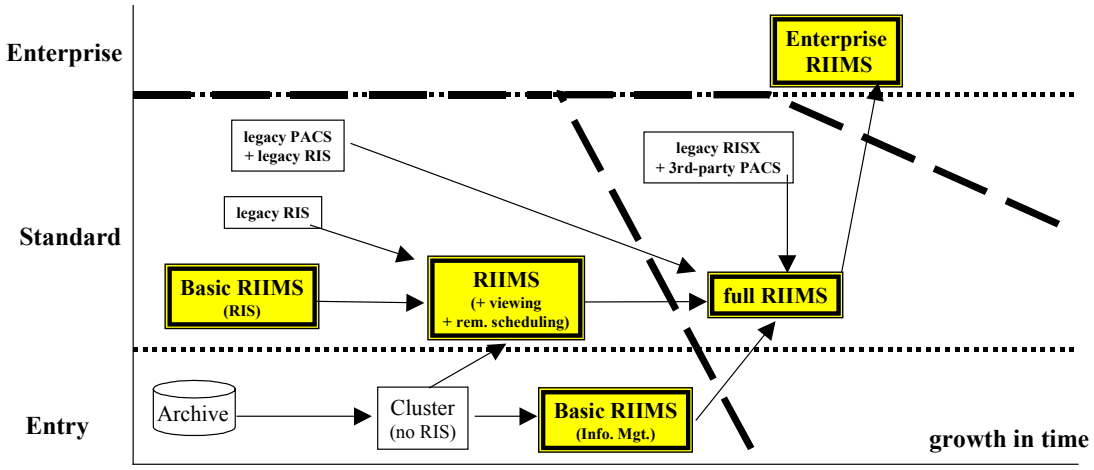


Figure 5.14 : Sample migration-map showing the existing and planned family members and the required migration/extension between them

Limitations

The ***migration-map*** does not show the variability *within* a particular member e.g. the features supported, but this can be derived from the appropriate ***family-feature-map***. It forces technical considerations to the business-level, but there is no guarantee (as with other tools) that things may not change.

5.5.4 Change-case-template

Definition

The change cases are the primary means to represent requirements in the method, and the template is used to structure the change-case. The template provided by the method will be described below with an example

Role

The template aids the stakeholder in expressing interoperability/extensibility requirements in a form suitable for assessment.

Who

The stakeholder uses the template to specify the change-cases and the ***architect*** uses it to extract relevant information.

The Template and Example

This is a (detailed) interoperability example based on a medical imaging system

Change Case <Name of change-case – name should capture the intention/goal.>
Provide handling of UltraSound(US) Images and Measurement data by brand-X PACS

Stakeholder: <The stakeholder responsible for authoring the change-case.>
XXXXXXX

Actor: < the parties (person/other-system) affected by the change – in case of an internal-system issue this may not be always applicable.>

ultrasound modality

Motivation:

Trigger: < The stimulus for the change (i.e. who, what initiates the change) – again in case of an internal-system issue this may not be applicable.>

US Images and related measurement data from the US modality – they shall be accommodated

Rationale: <Short description of the business case (preferably with reference to requirements, strategy documents) motivating the change – if different from "trigger". Business metrics – e.g. frequency of use, size of customer-base, priority, relation to core-business.>

More and more US modality vendors are confronted with (users) requirements to offer PACS based solutions in their product portfolio, optimised for US (colour) images and other measurement data. There a two business opportunities for the brand-X PACS in this domain;

- a dedicated US Image Server/Archive for the (larger) ultrasound lab
- a versatile Radiology PACS, supporting multi-modality imaging devices; CT, MR, XA, and US.

Specification:

Brief Description: <a brief description of the stakeholder-system interaction/behaviour that arises as a result of the change.>

Brand-X PACS (hereafter called the PACS system) should be able to receive, store, process and display (colour) image data and related measurements (results) from the US modalities. US studies may consist of US (colour) images, both ***single frame*** and ***multiple frame*** series. These images are transferred in a lossless of lossy (RLE) type of compression mode, dependant of the type of study, and may be accompanied with US measurement results, as well as ECG (curve) data.
The PACS system should be able to handle and display these (colour) images and measurements, so that the (primary) user of the system can review and manipulate these images on the systems (diagnostic) viewstation. Dedicated US practitioners may want to use a colour-based viewstations and combine these images with other type of examinations (e.g. CT or MR); other radiologist may want to use their high-resolution (B/W) diagnostic viewstation instead.

In addition, US examiners may want to review the dynamic behaviour of the US (multiple frame) images in a 'cine' loop type display; simultaneously looking at the moving ECG (curve).

Interoperability Context: <This is the context in which the change-case occurs. The environment both in terms of the system (the family-member) and its surroundings should be described with relevant detail. In particular: System topology (i.e. relationship with other systems, users); System settings (any special configurations, the particular family-members involved); Pre-conditions (necessary resources or conditions needed before the change can occur).>

The system shall support US images according the ***DICOM standard***; both single frame and multiple frames (cine-view) are to be supported. Large US cases may be transferred in a ***lossy-compressed*** mode, using the ***RLE compression method***.

A typical US examination case consists of patient/examination information, a set of US Images, measurement results, ECG data/curves (the latter in case of cardiac studies) and an audio (speech) file.

The PACS system should accept these types of images and multi-media information as a complete 'acquired image' set and stores all these images and information objects accordingly.

The PACS system should allow dedicated (colour) viewstations to be optimised for reviewing these US studies, making use of all the US (image) object elements as described above (inclusive speech replay).

In essence, all US images and other information are shown on these viewing stations 'as last seen' on the US modality.

Moreover, these US viewstations should be able to select one or more US image out of a examination series, add patient demographics and measurements data to it and to store the resultant image, called ***photofile image*** into the systems database for later reference or clinical reporting or teaching purposes.

These photofile images have then become part of the patient's (history) folder and it should be able to make a ***video hardcopy*** and/or export the photofile image(s) to an external DICOM workstation.

Other PACS-based (B/W) workstations may present these US images 'as best as possible'; restrictions on the US image presentation, image manipulations and the speech replay function may apply though.

Interoperability Details: <Any more detailed system behaviour required to support the requirement – sample illustrations of the desired behaviour are especially useful.>

see DICOM std. on US.

End note: <any notable conditions, circumstances for the system stakeholder after the case has been implemented. In particular if any other change-cases are affected (triggered, disabled).>

Assuming the US requirements as stated above are implemented in the system, another Change Case may be considered to facilitate the system for dedicated ***IVUS (IntraVascular UltraSound)*** studies.

IVUS studies are specialised examination procedures, performed on a Vascular X-ray modality with an add-on US acquisition device. A typical IVUS study consists of a series of XA images and US images that are correlated in the space domain (i.e. the X, Y, and Z coordinates of the two image planes are transferred with the images). Based on these (XA+US) image data sets, a three-dimensional, computerised graph of the vessel structure can be reproduced.

The PACS system should be prepared to support the storage of these IVUS examinations, and accommodate the reproduction and display of the vessel structures on the US (colour) viewstations.

Limitations

This is not intended to be a complete development-ready specification of detailed new/modified system behaviour. It is summary in nature and provides enough information for architectural assessment; the required level of detail is determined by the stakeholder and the architect. The intention of the template is to provide organisation and inspiration in specifying the requirement.

5.5.5 Change-case-guidelines

Definition

The ***change-case-guidelines*** are used to help the stakeholders in generating their requirements. This is particularly important for strategic stakeholders and system-quality requirements because of the immaturity of method support for these participants and development aspects. The guidelines for interoperability are included as examples below.

Role

Requirements generation is the key input to the assessment and supporting the stakeholders in this phase is the main focus of the method developed in this research. These guidelines are used to prompt the stakeholders to consider *family* aspects and also relate the other techniques (e.g. topology diagram) to the requirements-generation process.

Who

The **facilitator** guides the ***stakeholders*** during brainstorming to answer these guideline questions for their particular family.

Example

Following is the initial list of useful questions/checklists that emerged when discussing ***interoperability***:

1 Do you know (list) the external systems that could/will co-operate with our candidate system?
2 Do you know external systems topology (the ***family-context-diagram***) ?
3 Do you know the variances within an individual system-class
4 How do you distinguish between system variants
5 How many classes of external system exist?
6 How many instances of each class typically must be supported (synchronously, asynchronously)?
7 How does external-system variance impact our system?
8 How do we account for these impacts (at architecture level?)

These questions can be used in each case of interoperability requirements/architecture description in order to both initiate and structure relevant discussion.

Limitation

This set of questions is not exhaustive, but it is a solid start on which to base ongoing interoperability understanding and organisational learning.

5.5.6 Requirements-ranking-criteria

Definition

The ***requirements-ranking-criteria*** are important in differentiating the importance of the various change-cases. The FAAM ranking criteria are provided as examples below.

Role

Ranking of change-cases between stakeholders is important where selection must be done because of time constraints in assessments. The criteria can be used to select the 5-9 most important change-cases (steps 2,4) prior to assessment and can be also used during the assessment meeting (step 5) to divide time appropriate to the importance of the change-case.

Who

The stakeholders as a group should rank the competing change-cases, in case of disputes organisation culture will dictate whether stakeholders should have equal or weighted votes or whether the business-manager should over-rule

Example

In order to rank competing change-cases in importance the following heuristics will prove useful:

- Change-cases with largest *business impact* for the producer customer should be ranked highest (**metric** e.g. frequency of change-cases, customer-base affected, priority of associated business strategy)
- Change-cases with largest *impact on the architecture* should be ranked highest (**metric** e.g. number, importance of components affected). **Note**: this metric is typically only known *after* an assessment, and is important in highlighting architectural *weak-spots* and in determining re-architecting priorities.
- Change-cases with most impact on the external system environment (**metric** e.g. number, criticality of external systems/components involved) should be ranked highest, especially when addressing interoperability.

Limitations

This is the initial ranking based on the business and technical drivers underlying family architecting. They provide an initial guidance to participants in selecting between change-cases and may be extended by individual organisation expertise.

5.5.7 Assessment criteria

Definition

The ***assessment criteria*** are extra (to the change-case descriptions) requirements properties important to the success of the assessment. Assessment criteria are set *before* the assessment execution in order to ensure that as rational, and objective an assessment as possible is enabled.

Role

While satisfaction of the quality-specific change-cases is the main assessment criterion, the assessment criteria serve to provide qualifiers to the change-cases to aid in the effectiveness of their assessment. Extra assessment criteria from FAAM are provided as examples below.

Who

The criteria must be developed with input from the stakeholders and explained to the assessment team prior to the execution.

Example

Some extra criteria that will increase the effectiveness of the assessment are identifying:

- those changes-cases that **must** be totally supported (will avail of ***requirements-ranking-criteria*** provided earlier);
- those change-cases where there is some room for negotiation, and the negotiation-limits;
- (un)acceptable architectural-change costs (link to business value).

These criteria are used:

- in the prepare requirements/architecture activities to ensure that the most important issues will be addressed during the meeting;
- during the conformance assessment meeting to focus the discussion;
- during the conformance reporting activity to indicate where re-architecting or defeaturing priorities should lie.

Limitations

These are initial proposed criteria, and may be used or extended by the development team as appropriate to individual circumstances and the formality of the review.

5.5.8 4+1 architectural views

Definition

The ***4+1*** approach of Kruchten [1995] towards representing architecture is adopted as good-practice in this assessment method. The 4+1 has been discussed in Chapters 3 and 4, and comprises:

- the **logical** view – describing the design's object model or entity relationship model;
- the **process** view – describing the design's concurrency and synchronisation aspects;
- the **physical** view – describing the mapping of the software onto the hardware;
- the **development** view – describing the software's static organisation in its development environment;
- the **use-case** view – describing how the elements of the other views work together to provide important functionality and features.

It is representative of the multiple-view approach to architecture description accepted in the broad research community, and many of the other approaches to multiple views can be mapped directly to elements, or to combinations of elements, of the 4+1 [Kruchten, 1995]. Further the 4+1 approached is embedded (indeed it originates) in the development method advocated and supported by the current leading CASE-tool vendor.

Role

The ***4+1*** **view** provides a means for the family architecture to be presented for assessment. The architecture must be communicated to the stakeholders and it must be shown to satisfy the requirements, so a representation is necessary – the value of the 4+1 approach is in

providing some attempt to standardise the number and focus of such representations. As stated earlier, the 4+1 is advised, but not mandated by this method.

Who

The ***architect*** usually determines the views and notations he chooses for representing the architecture. The ***facilitator*** can use the 4+1 as a reference model to monitor progress in architecture description.

Example

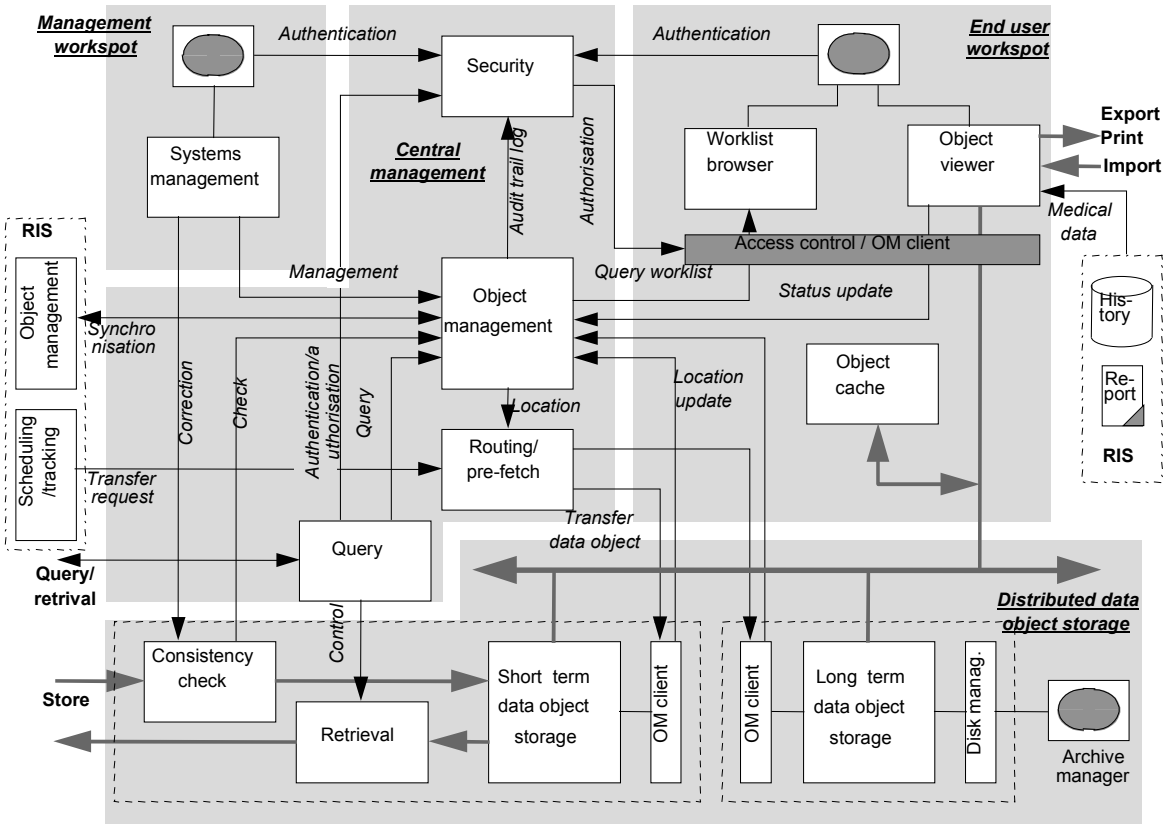


Figure 5.15: combined logical and deployment view of family architecture

Legend: This combined architecture view is typical of those found in practice in that it is described using a home-grown notation, and addresses multiple concerns.
components – logical services (framed rectangles), and deployment resources (shaded, unframed rectangles)
connectors – data flows (thick lines); control flows (thin lines)

Limitations

There are many ways to represent architecture, and the various representations have grown-up within organisations and have served them well. Not all of the 4 views are needed for every architecture [Kruchten, 1995], and some subset or combination may be appropriate for individual assessments. The practice of architecture assessment should not be blocked because of organisational resistance to change its architectural notation.

5.5.9 Requirements-architecture interaction-metric (from SAAM)

Definition

The ***req.-arch.-interaction-metric*** describes per *change-case*, which architectural components are affected in supporting the change.

Role

It provides a tabular summary of the architectures satisfaction of the requirements. It can also briefly summarise the details and costs of these changes.

Who

The metric can be drafted by the ***architect*** prior to the conformance meeting (step 5) and completed after that meeting when the stakeholder feedback has occurred. It forms part of the ***conformance-statement***.

Example

CASE 1 (see example in section 5.5.4):

Architecture components impacted (see Figure 5.15):

- lossless storage – long/short-term data object storage (2)
- lossy storage – long/short-term data object storage + consistency-check (3)
- image display – none (0)
- B/W image + data display – object viewer (1)
- colour image + data display – as for B/W + special H/W

Magnitude of impact

4 components – most change (90%) in building application for MM display

5.5.10 Sensititvity-metric (from SAAM)

Definition

The ***sensitivity-metric*** lists per *architecture component* the number of change-cases affecting it.

Role

It provides a tabular summary of the sensitivity of each architecture component to the requirements. In so doing it can give an idea of how modular the functional allocation has been or whether a more fine-grained representation is needed for the assessment.

Who

As with the previous technique, the metric can be drafted by the ***architect*** prior to the conformance meeting (step 5) and completed after that meeting when the stakeholder feedback has occurred. It also forms part of the ***conformance-statement***.

Example

See Appendix B and C for robust example

5.5.11 Conformance-statement

Definition

The ***conformance-statement*** describes, per change-case, how the architecture addressed the requirements and what the consequences and actions should be.

Role

It is a technique (document) to officially record and communicate the architecture's performance in the assessment to the stakeholders and management team. It is based on briefly describing the assessment findings according to the following template:

1. **Introduction**
2. **Assessment context and aims**
3. **Requirements overview**
   3.1. Prioritised change-case specifications and assessment criteria
4. **Architecture conformance**
   4.1. General architecture overview
   4.2. Per change-case - how (with brief reference to views) the architecture has addressed the requirement
5. **Indications of analysis (costs, risks, follow-up actions)**
   5.1. Uses interaction and sensitivity metrics
6. **List of decision issues for management with costs and recommendations from assessment team**

This document captures the knowledge generated by the assessment and, via the recommendations, is a starting point for incremental architecture and assessment improvement.

Who

The conformance findings are jointly prepared by the ***facilitator*** and ***architect*** (to ensure technical clarity and establish ownership). The information is communicated back to the ***assessment sponsor*** and the ***stakeholders***.

Example

Some of the conformance-statement items (***req.-arch.-interaction-metric***, and ***sensitivity-metric***) have been described already. Due to the confidential nature of this artefact an example will not be provided.

Limitation

These statements are broadly organised as depicted here, but may be customised to reflect individual concerns; and extra details (e.g. business motivation behind family, reason for assessment, may be appended as required.

5.6 Reflection: contributions and constraints

A step-by-step process for iterative, stakeholder-centric assessment of an information-system family architecture for interoperability and extensibility has been described. Eleven techniques have been presented to aid the development team in assessing their architecture,

these techniques have been defined, their role in the assessment process explained, and their usage illustrated by industrial example. The techniques, the main contribution of the research, have concentrated on supporting family stakeholders in generating, and structuring their requirements. The method and techniques have been developed in accordance with design principles outlined at the start of this chapter, which in turn are motivated by the requirements derived in the problem analysis from Chapter 4. The method process and the techniques described above have comprehensively answered the remaining two research questions: how to assess the architecture? and how to communicate the results?, respectively.

The following paragraphs highlight the key contributions and discriminators of FAAM compared to other approaches, especially SAAM

Contributions

- **Stakeholder-centric throughout**. Stakeholders are not simply "polled" for requirements, which are then taken over and re-interpreted by architects and used to guide the other assessment activities. In FAAM stakeholders are empowered to fulfil their responsibility; to assume ownership for their requirements and to maintain responsibility for their refinement and assessment.
- **Architecture assessment is incremental, repetitive, and "normal"**. The *individual* generation of change-cases by stakeholders is a marked departure from SAAM; and recognises that in the incremental self-assessment approach of FAAM, architecture assessment must be seen as a normal development activity where individual preparation is typical. The effectiveness of individual requirements generation prior to group discussion for architecture assessment has (independently) been experimentally validated elsewhere [Bengtsson and Bosch, 2000]. The perspective of ongoing assessment underlying the method also means that the change-case reduction actions described in step 1.6 of the method do not have to be so exclusive. Important change-cases that do not make the "top ten" are not lost, but become candidates for repeated iterations of the assessment-step (step 5). This means that multiple assessment meetings can be prepared in a single change-case generation and ranking meetings. This flexibility is attractive where people would like to batch the "requirements-generation" part of the method due to scheduling economies or related project milestones.
- **Architecture assessment as a repeatable process**. The main motivation to FAAM is providing organisations with some techniques to adopt the approach initiated in SAAM for themselves. This means they acquire ownership of architecture assessment practice and the quality improvements it enables becomes an organic part of their process. The provision of techniques and tools to enable a repeatable, *expert-independent* process is necessary for widespread industrial adoption. FAAM is an initial attempt to bring architecture assessment to (in CMM terms!) a level 3 (repeatable) status and is a key contribution of this research to SAAM.
- **Simple "how-to" methods to start, extend later**. The approach to supporting industrial users in applying the FAAM process is to provide some simple starter-kit techniques.

This lowers the entry barrier towards adoption, which is important, as starting is often the most significant action in process improvement. The method has outlined the key issues to be addressed in architecture assessment and their solution design in section 5.4.2. The techniques implemented in section 5.5 are an *instance* of the designs. These proto-techniques can be extended as organisation experience with assessment matures and as more-powerful assessment techniques emerge from research. In fact the design rationale described in section 5.4.2 has indicated advanced work in several associated areas e.g. [Kazman, *et al*., 2000], [DeBaud and Schmid, 1999], [Bosch, 2000], [Lassing, *et al*., 2000].

- **Family focus**. The method takes a family focus, both in identifying the stakeholders and providing guidelines on their concerns. The qualities chosen for technique development (i.e. interoperability and extensibility) are particularly relevant for families, and the concepts of intra-, and extra-family aspects to these "established" qualities has been developed (see Chapter 4) and their practical application supported by simple techniques.

As with all design, however, there are some aspects of the problem which have been excluded from the method and techniques and it is important to make this explicit.

Constraints:

- **The method ignores certain aspects of interoperability and extensibility** (e.g. performance, low-level technical protocols, executable architectures) – the reason for this is that these aspects are *secondary* to the main concern of "What interoperability are we actually going to support?". This is not to say that an important topic like performance is unimportant – it obviously is and there is a significant performance related aspect to interoperability. However, as noted elsewhere (e.g. [Abowd, e*t al.*, 1997]) performance aspects are best suited to a non-paper, simulation or prototyping review. Further there is a natural progression in architecting, and typically the *logical* view is more established earlier in the architecture life cycle. Addressing performance issues typically follows the initial identification and understanding of the interoperability requirements as discussed in this method.
- **What about peer-review by other architects**, who can best appreciate architecture descriptions and the architecting task? This issue has been addressed in Information Box 4-A, and is outside the scope of this method – which is explicitly dedicated to stakeholder-based assessment. The position taken is that peer-review by architects is best done at a later stage to optimise the technical aspects of the architecture, after the stakeholders are satisfied that the right problems are being addressed. Methods such as SARB [Weiss and Lai, 1999] and those described by Bosch [2000] are appropriate complements to FAAM.
- **How does it relate to the ongoing work in the SAAM and ATAM methods**? This method and the accompanying techniques provide a necessary how-to supplement to SAAM, to facilitate its application in organisations by organisations (the do-it-yourself

approach), to address family issues and to address interoperability and extensibility. ATAM is a more advanced method that SAAM and recognises the multi-criteria trade-offs inherent in architecture development. The FAAM method, like SAAM, can be used as a single repetitive unit in ATAM's cyclic architecture assessment programme. The approach here mirrors that of SAAM/ATAM development: first a process has been defined for a single-quality perspective on assessment (SAAM), later the multi-quality dependencies are supported (ATAM). FAAM is adding a family-perspective and practical techniques to SAAM, incorporating family issues and providing more advanced techniques to further exploit ATAM is work for another day (researcher).

5.7 Summary

The principles underlying the design of the FAAM assessment method have been listed. These have been applied to the problem area and the assessment method has been described in terms of the process steps that should be followed. Additionally, in contrast with most assessment methods, practical advice is provided as to what *general* additional activities are needed to provide the information-artefacts required to fulfil the process steps. In additional to practical advice; eleven techniques and metrics have been provided and their use by participants described. Examples include the ***family-feature-map*** and ***change-case-template***, which provide representations to aid stakeholder communication and therefore the assessment process.

The following chapter describes the application of FAAM in two industrial case studies to explore its application to the system qualities of interoperability and extensibility. The intention is to validate/extend the general method and also to begin initial development of the quality-specific aspects of the method.

Chapter 6
Implementation – case study application

6.1 Purpose

The purpose of this chapter is to:
1. report on the application of the method and techniques in an industrial setting;
2. extend the method and techniques based on the lessons learned;
3. evaluate the usefulness of the method from a practical standpoint.

This is done based on two case study applications of the research products (method and accompanying techniques presented in Chapter 5).

As discussed in Chapter 2, the case study is the primary means to validate the research results against the claims made there, and detailed in Chapter 4. Validity is strongly based on logical argument and the provision of generalisable design knowledge [van der Zwaan, 1998] supported by the case-study observations. The case-study implementations are additionally used to provide feedback on the usefulness and usability of the method, so that the design is enriched through its incremental application in a real setting. It is expected that method refinement (particularly with respect to its inclusion in the "normal" development process) will be ongoing as more such application experience is built up outside the confines of this particular research effort. These dual aspects of the case-study experiences will be treated separately in the analysis of the results.

In section 4 of Chapter 2 the intended contribution of the research was claimed as:
1. explicit attention to, and *active* involvement of, product-family *stakeholders*
2. focus on a selection of family-relevant *system qualities* (interoperability and extensibility)
3. emphasis on practical "*how-to*" mechanisms and *techniques* to enable the development teams within the producing organisations to implement the method.

The feedback gathered from the cases is used to evaluate how the research performed with respect to the above intentions, and is categorised as follows:
- the success of the method with respect to **assessing** family **architecture** – architecture assessment is the key theme in the research and this provides some general feedback on the assessment experiences;
- the inclusion of **family** aspects in assessment (i.e. stakeholders and the qualities of interoperability and extensibility) – this is a significant extension of existing practice,

exploring interoperability and extensibility from a family context (see contributions 1 and 2 above);

- the effectiveness and enhancements of the practical **techniques** provided by the method – the provision of techniques enabling self-assessment by the development team is the main contribution of the research towards improving the practical applicability and increased exploitation of architecture assessment of information system families (see research contribution 3 above);
- general **implementation guidelines** for applying the method in and by organisations – the practical focus of the research warrants the attention paid to getting the method introduced into the socio-political context of product-family development This complements the previous contribution and also addresses one of the secondary research areas indicated in section 5 of Chapter 2, i.e. encouraging architects to open-up their architectures for assessment.

6.2 Structure

The chapter begins with a general discussion on the methodological context of the case study as a validation mechanism. The practical focus of these particular case-study validations is also presented. Following, the design rationale associated with the case selection and implementation is explained, i.e. what are their characteristics?, what particular aspects of the method are being explored?, and what constraints are imposed? (what trade-offs arise, what is not being explored). Some time will be spent discussing how feedback was gathered, and what measures (quantitative and qualitative) were performed. The case study profiles are provided illustrating the common and varying aspects between the cases and their relationship to the validation exercise. Hereafter the individual case study implementations and associated method experience is presented, including reference to actual case material the details of which are included in Appendix A. The experiences per case are analysed and general lessons with respect to validating, and extending the method presented.

In concluding the chapter, an overview of the method experiences is provided based on the categorisation presented in section 6.1 above; and those lessons learned with respect to the applicability of the method recounted. In summarising the implementation findings are compared to the claims made for the method in Chapter 2, and some interesting observations are highlighted

6.3 Method operationalisation and validation with case studies

Case studies in co-operation with the techniques developed for analysing the system qualities of interoperability and extensibility are used to operationalise the method. Industrial operationalisation is a key theme in any engineering research and of special interest to this research as one of the main aims is to provide a means for family development teams to assess their own architecture as a means of improving their collective architectural practice.

Industrial usefulness, manifest by the experience and opinion of method participants, is therefore an important part of the research validation. For this reason, considerable weight will be given not just to the absolute results of the method in terms of concrete architectural-quality artefacts – but also to the process of applying the method and the participants' involvement therein. In this respect the research validation actively addresses the transferability of the research ideas to industrial practice.

As discussed in Chapter 2 – the case study approach is a suitable research technique for "how" and "why" research questions concerning a contemporary set of events where the investigator has little or no control (in the experimental sense)[1]. [Yin, 1994]. The research questions proposed here (how to assess architectures) and the application area (information system family development) clearly match the previous criteria.

The case study design (see section 6.4) describes the scope, structure and interpretation logic associated with the selection and implementation of the case studies. The case study design approach in this research deals with a small number (2) of cases. As indicated by [Yin, 1994] ***multiple cases*** are chosen to provide a more robust research observation set. This relies (as does all case study research) on "analytic generalisation" – where similar, or contradictory, results arise across the cases according to the rational prediction of the research theory – to establish the broader applicability of the findings. This is in contrast to *statistical generalisation*, where the focus is on the generalisability of a sample to the whole population.

Analytic generalisation aside, the research recognises the importance of repeated industrial exposure and this would have been pursued to a greater extent had resources and time been available. In order to support such extensive validation, the method implementation has been described in sufficient detail to make repetition in other contexts and by other researchers easier, and comparable.

6.4 Case study design

The overall aim of the research is to provide a method that information-system family developers can use to assess their architecture for conformance to the system-quality requirements of interoperability and extensibility. The primary aim of the case studies is to validate the method and techniques proposed for assessing the architecture by the development organisation[2]. The validation is twofold, addressing the main aims of the research:

1. Does the method successfully address the inclusion of *family* stakeholders and the family concerns of interoperability and extensibility in architecture assessment?;

[1] Brackets are this authors

[2] This is not simply the development department – but the aggregate organisation responsible for the creation and support of the system (family); including product management, marketing, and maintenance.

2. Are the "*how-to*" techniques provided by the method of practical use to the participants in implementing the method?

In providing the validation – certain aspects/entities associated with the cases must be analysed. Due to the socio-technical nature of the research area, there are multiple interacting parties and processes, and hence the case study analysis is concerned with multiple analysis objects; this is termed an *embedded* case study [Yin, 1994]. The analysis units are:

- Main unit of analysis
  - The method
- Sub-units of analysis
  - The method process
  - The architecture
  - The other method artefacts
  - The participants as a group
  - The participants as individuals (architect, stakeholder)

In concrete terms this means that case results and experiences will be gathered from the above entities. How these entities are investigated for results is dealt with later in a dedicated measurement section.

6.4.1 Case procedure

In each case study the case and its context is briefly described, highlighting any important (for the method) constraints or conditions. A detailed report is provided in Appendix A as to how the method was applied in the case. This report will follow the method process steps (see Chapter 5, and Figure 6.1 below), as this is how the case was implemented. Additionally, the detailed activity diagrams for each of the method steps will also be discussed per case, especially any deviations from the proposed design with their motivation and results. Observations and lessons will be provided in the concluding section of each case report. More details of the method can be found in Chapter 5, describing the method design.

The close coupling between the method definition, data gathering, and reporting from the cases means that detailed feedback on individual method aspects is possible (this is analogous to component testing in system development). The concluding step where the participants and the facilitator (researcher) reflect on the method as a whole is used to generate a method-wide overview evaluation (analogous to the system test).

6.4.2 Case study constraints

There are also some practical issues to be borne in mind when doing industrial-context case studies. The two main challenges in doing such method-, process-related case studies are:

1. the challenge of *introducing* the new way of working (the proposed method) into the case environment (by definition a real, running organisation);
2. the normal research challenge of measuring the effectiveness of the method as *finally implemented* in the organisation.

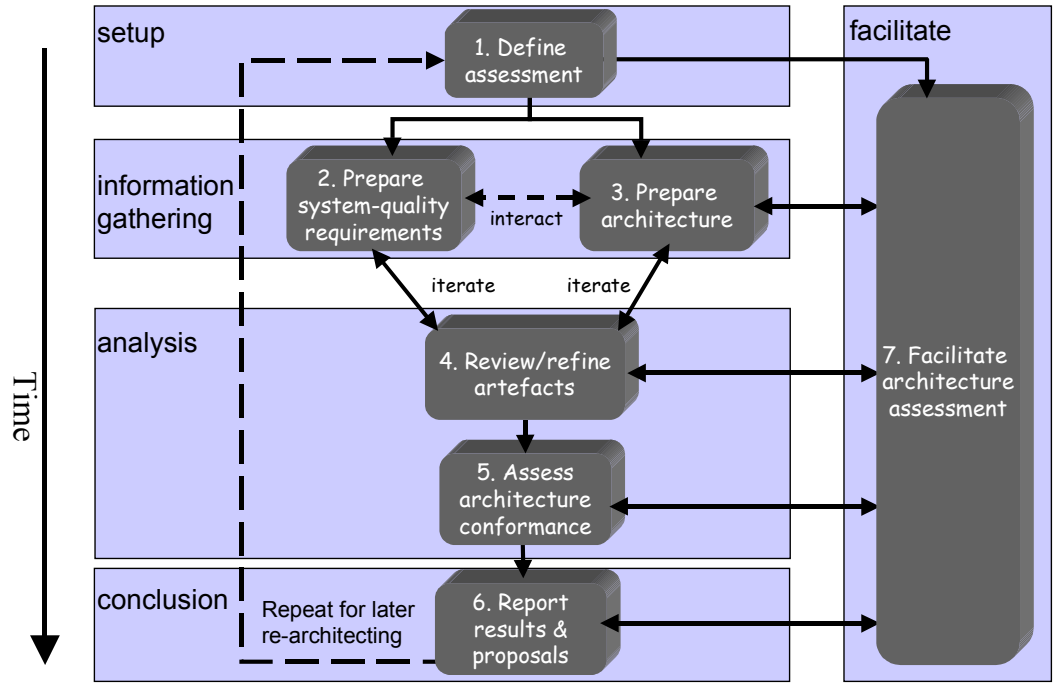


Figure 6.1: Overall FAAM process

Typical consequences of these challenges are:

- because of the learning effect necessary to introduce methods, only *partial* method introduction is possible in initial cases – i.e. you cannot test the full scope of the method (termed here the breadth problem – not everything is used);
- because of organisational variances and local effects, implementation will deviate from initial design, and from other implementations, and method steps; techniques will not be used in the exactly intended/prescribed way (termed here the depth problem – what's used is not used exactly as intended[3])

The consequences of these challenges for research validation – is that there will necessarily be some compromises between the full validation demands of the research and the possibilities offered by the real industrial setting. The role of research design is to manage and mitigate the compromises, and the role of reporting is to truthfully reflect these compromises and indicate their effects on research claims. Both these strategies are pursued, and therefore the reader can expect some deviations between the design proposed in the previous chapter and the actual implemented experiences reported in this chapter.

[3] Having said this it is important to note that this is no more than a fact of life when introducing something new to almost any complex system. For example, most people can tolerate food additives – but a certain number will develop unpredictable allergic reactions, because of certain individual conditions prevalent at the time and the precise way they metabolise the additive. The nature of software systems development, and the case study is even less amenable to the extensive predictive theory and trials associated with medicine (there is no 3 thousand year old professional body of knowledge for software). The key thing to remember is that the initial aim is to extract the general picture, we are not at the detail clinical trials stage but are still in the lab; addressing the issues, and the potential to scale up. This is appropriate at this stage of any systems research.

6.4.3 Case selection and description

The cases selected are termed the *medical case* and the *planning case*; the names are based on the application domains addressed by the respective systems.

The case studies were primarily selected based on aspects that the cases had in common, both with each other and of course with the research area. Both concern software intensive systems, both are produced by relatively large parent organisations (> 4000 employees), although the individual business units are medium sized (< 200 employees). Both businesses are Dutch based but have a strong international market presence, and orientation. In addition the system families under consideration are traded and supported internationally, therefore there is a wide scope to the families as they deal with international variation. Both organisations are process-improvement *positive* – in the sense that they are either actively pursuing internally-driven improvement programs such as CMM (both cases), or have a proven track-record and ongoing interest in externally oriented national and European-wide research partnerships and projects (the planning case). Most importantly both organisations are actively orienting themselves towards product family based competition – but these efforts are ongoing and have not matured in the sense that they have not yet fielded multiple generations of multiple product lines, based on a platform of reusable components. Hence they are still interested in efforts to better support product-family development.

The cases have also some significant differences – best illustrated by presenting the individual case profiles.

6.4.3.1 Medical case – profile

The medical case concerns a product for viewing and manipulating radiology images to add clinical value. Although the images may be later distributed by other systems outside the departmental boundary – the added value of the system is mainly within the department in supporting the primary tasks of the radiology professional. With respect to the particular project chosen – it is a small (< 3 man-year) pre-development (proof-of-concept rather than market-release) project, aimed at providing a common-viewing component to an existing suite of related image-processing applications. The project was successfully completed in summer 2000, with concrete plans to incorporate the proven concept in the commercial product family.

At assessment time the project was almost complete and the team was preparing for the productisation of the findings, with the necessary extensions this entails. The team saw the assessment mainly as a chance to prepare for the productisation by looking towards its *extension* to deal with family variety both from a requirements and architecture perspective. Although the project is pre-development – it makes extensive reuse of existing (fielded) platform services; in this sense the assessment can be seen as architecture-*validation* as

portions of the current architecture are fixed, while the requirements are expanding. See Table 6.1 for a summary of the medical case.

6.4.3.2 Medical case – family aspects

The medical case deals with a *common* viewing *component* and its associated architecture and interfaces with existing components in the image workstation *family*-suite [Jacobson, *et al*., 1997]. The aim is to replace 3 existing viewing systems and provide their (viewing) functionality in a *single common component.* This must work with other *family assets*, including the existing application *platform*; so that customers see the smooth *evolution* of the system. Further the component *extends* the current functionality by offering *user-customisable* settings and workflow

Information Box 6-A: Pre-development – what does this mean ?

Pre-development projects differ from market-release projects in the sense that their results are not intended to be sold, installed and supported for end-customers. They do, however, typically provide some concrete and important deliverables for internal company use.

As the name suggests, these projects *pre*-empt the larger *development* effort of market-release projects; and usually the success of the pre-development project is determining on the inception, or scope of the market-release project. In concrete terms pre-development projects are a pro-active means to reduce risk prior to committing to a "real" market-release project. In most cases these projects seek to reduce risk associated with *technical* aspects of development, e.g. by acquiring knowledge, and/or proving the domain-applicability of a new technology, often making extensive use of prototyping. They are, therefore, an ideal context in which to implement the risk-reduction measures motivating architecture assessment and the assessments can often provide information to guide (or even initiate the need for) prototyping efforts.

Pre-development projects are usually of small-scale (e.g. <6 man-years) and are not as heavily managed as market-release projects. This does not mean that they are unimportant however, in contrast, they are typically strategically focused and their output will be used by business management in deciding on how to proceed with the associated market-release projects. In the cases described here the pre-development approach is used to reduce the risk associated with embarking on a family-development model. This is also true elsewhere, and the wisdom of pre-development as a necessary step in family development is becoming increasingly recognised (see e.g. [Bass *et al.,* 1999] pp13), and it seen as integral to the software family business. They are the "greenhouse" used to develop (with proof) the requirements and/or design necessary before embarking on larger investments needed for the market-release project. The projects reported herein delivered (or plan to deliver) approx. 80% of the final market-release design, and therefore limit the risk of these projects to providing an efficient implementation.

All this means that the positioning of the case studies as pre-development projects:
does not in any way limit their applicability to the "real" business of bringing family-members to the market;

explicitly conveys the fact that development of critical family-assets (e.g. requirements, architectures, common-components) is typically done in incremental, proof-of-concept mode before scaling up to the remaining organisation and the market.

6.4.3.3 Planning case – profile

The planning case concerns a large (see estimates for full-scale development effort in Table 6.1) highly configurable software product, which covers all the primary operational business processes including ordering, purchasing, production planning, delivery and invoicing. The overall product's scope is very wide, crossing multiple departmental boundaries even crossing single organisational boundaries. This particular project is a pre-development project, aimed at *re-engineering an existing system* to make it more amenable to family development through separating the existing applications into loosely coupled components, thus providing the configuration flexibility typically associated with families. The short-term business motivation for separating components is to support multi-site business processes – the part of the project focused on in the assessment was the modelling and communication infrastructure to support component *interoperability*.

The project is in design phase (no implementation has been done) and at assessment-time the architecture is early in its life cycle with the logical view dominating. For these reasons the prime aim of the assessment was to:

1 Verify that the *current* architectural concepts support the multi-site business requirements as perceived by the stakeholders;
2 Use the requirements to generate insight into a *further refinement* (sub-components) of the architecture, in particular the intention of the sponsor was to use the assessment to generate input regarding what must be subsequently prototyped to prove the architecture.

In this sense the assessment is more *architecture discovery* rather than certification of a "complete" architecture against a fixed set of requirements. See Table 6.1 for a summary of the planning case.

6.4.3.4 Planning case – family aspects

The planning case refers to the re-engineering of a large *customisable* system (see [Jacobson, *et al*., 1997]) providing a *wide variety* of logistics functionality. The specific aspect explored in the assessment is the introduction of a new *common* communication infrastructure. This will support more *decoupled*, application-component *interoperability*; thus enabling *modular* application components to be developed, and easily *configured to customer wishes*; and also third-party components to be integrated. Further these infrastructure components must themselves be *replaceable* independently of the (*version* of the) other infrastructure-, and application-components.

Based on the overview, the cases provide evidence of the method's application with some specific context settings. These settings provide limits to the implementation of the case, and

perhaps to the validity and scope of the research claims. These issues are presented here, they will be returned to both in case reporting, and in the concluding analysis at the end of the chapter.

Table 6.1: summary of important aspects of case-profiles (differences shaded)

Attribute	Medical case	Planning case
Context	Department	Enterprise
Users – No. / range	Few – Small	Many – Large
System size	Medium	Large
Market type	Business	Business
Application domain	Medical imaging	Enterprise Information Systems
Family maturity	Immature	Immature
Architecture assessment Experience	None	None
Est. Project size (pre-development size)	10's of MY (3 MY)	10's – 100 of MY (4MY)
Project type	Pre-development	Pre-development
Project phase	[4]80% complete	30% complete
Family aspect	Common component	new communication-platform architecture
Family typology[5]	Suite	Single application (variant)
Interoperability focus	✗	✔
Extensibility focus	✔	✗
Assessment type	Validation[6]	Discovery

6.4.4 Generalisation aspects

An important aspect of case study research, especially when validating claims of generality, is to have common threads typifying industrial practice unifying the cases and to build-in differences thereby exposing the method to a range of situations. This ensures that the cases are (analytically, not statistically) representative of the industrial application domain. Case study design has taken steps to ensure this, and the relevant case attributes are discussed here.

Common threads

Both cases provide evidence of method application under the following circumstances:

[4] Note this is a pre-development project, so the content restrictions may not be as severe as market projects.

[5] based on family typology advocated by [Jacobson, *et al*., 1997], **suite** of related-applications (MS office), **single** application customised in **variants** (Ericsson AXE telephone switch), **independent** applications based on similar components (MS foundation classes), see 3.4.1.1.

[6] Validation in the sense that the change-cases impacted re-used platform components providing services to the viewing component – this may be outside the scope of the then medical-case project but this platform is part of the surrounding family.

- In **family-creation** environments – this is useful as the architecture is an essential, constraining aspect of the family definition and attention to architecture here is crucial (see Chapter 3). This is the intended domain of the method and does not constrain its intended claims.
- In groups **without experience** of architecture assessment or family development – this is not a limiting constraint because it is still representative of the state of practice (see Chapter 3) of software development, where architecture and families are still relatively immature practices. Furthermore this lack of experience also represents a tougher test on the claims of the method – if it proves usable and useful in an inexperienced setting, then it is logical to assume that it will be still more useful in an experienced situation. The inexperience in the domain does suggest that the method should be re-evaluated, with groups that have built up experience, when industry has reached this level of maturity.
- In **pre-development** projects – this is also typical of companies who are starting out in family development, and want to crawl before they run (see Information Box 6-A). Although this may seem to limit the applicability of the method with respect to "real" market-release projects, the necessity for pre-development and the suitability of assessment at this time has been argued in the previous discussion. Furthermore:
  - installation of the method here is "seed-capital" for when the organisation leaves pre-development mode;
  - it's most likely (in fact it should be mandatory if real lessons are to be learned by the organisation) that the same people and processes responsible for the pre-development activities are included as key-people in the market-release project, where they themselves can scale-up and disseminate their assessment practice.

The position taken in this research is that there are, in principle, no problems to method claims for a market-release project. However, repeating assessments in market-release environments would have been a useful contribution to the research in terms of ensuring such validity.

- In projects that are dominantly **technology driven** – this is typical of pre-development projects which are frequently concerned with proving a technology concept. For both groups the assessment was used as a means to instil some more business/application focus to the project. However, because this was also the first such assessment for these groups, we confronted the breadth-problem meaning that not all aspects of the method could be exercised in the first go. This means that some of the most important method techniques (especially the more business-focused techniques such as the migration-map) were not exercised fully in both cases. Again, this is to be expected when introducing new techniques to a new audience in a new organisation. Repeat assessments with the same group will provide a chance to exercise these stakeholder-oriented techniques more fully. However, experiences to date – and hence claims – are limited.
- In groups that are **practice-improvement** oriented – meaning that the organisation culture supports changing and extending work-practices. This is a prerequisite for most case-study research validation. It is an interesting thought whether this is also a prerequisite for

the successful introduction of architecture assessment and perhaps even the adoption of product-family based development. This thought will not be explored further, and the method will make no claims with respect to improvement attitude of the company.

- In situations where **common** aspects of the family dominated – rather than those variable aspects. The evidence seems to indicate that identifying, and concentrating on producing the core/common components at the heart of the family is the most natural way to start family development. For the method-techniques this means that the family-map (concentrating on variety) may receive reduced attention.

Differences

Varying aspects that explore differences between the cases include:

- The focus in the medical-case was on **extensibility** of the current prototype and family functionality; whereas in the larger planning case the emphasis was on exploring how the key concept of component-**interoperability** would be addressed. This was by design, and means that both main quality-issues underlying the method techniques can be assessed. Although **not together** in the same case. The consequences of this will be revisited in the case study conclusions.
- Both cases differed in the size of the assessment team. The medical case was small and assessed completely by the core pre-development team using the internal stakeholders, the planning case involved a larger group of organisation-wide stakeholders, which tested the method's **scoping possibilities**. Further the applications domains and the system size are very different which again is useful to test the generality of the results.
- The cases deal with both **early-phase** (planning case) and **later-phase** (medical case) architecture life cycles. This means that the method can be explored both when the architecture is being *discovered*, and when it is being critically *validated* (see also Information Box 5-A).
- The fact that in the planning case, circumstances dictated that certain short cuts with the method implementation were taken. This provided the opportunity to use the "predicted failure of counter arguments" approach to case study validation [Yin, 1994]. This means that one deviates from the prescribed method, encounters problems and thereby induces that the prescribed method would have worked best. Such an approach was used in the planning case, where the attempt to introduce the method "lightly", by the architect describing the method process and techniques individually (instead of by the facilitator, in a group context), led to some stakeholders being unsure as to the process, and the use of some techniques.

6.4.5 Measurement in the case studies

Measurement is based on the units of analysis presented earlier:

- The method process;
- The architecture;
- The other method artefacts;
- The participants as a group;

- The participants as individuals (architect, stakeholder).

The method itself provides quantitative metrics on the concrete artefacts. These are supplemented with qualitative metrics based on feedback from the method participants. As the method is mainly focused on providing participants with more tools to do their job; this feedback is a valid form of measurement.

The cases are designed to provide a mix of people; the planning case, in particular, had a wide variety of stakeholders, and the fact that different companies are used also avoids any single-company bias.

6.5 Overall conclusions from case studies

In the interests of brevity, detailed reports of the individual case study implementations with accompanying analysis of results are contained in Appendix A. The remainder of this chapter will concentrate on presenting and analysing the joint findings across the case studies. The overall conclusions presented here are drawn from synthesising the appropriate lessons from the individual cases as reported in Appendix A, and structured according to the observation categories identified in section 6.1 as related to the main contributions of the research.

6.5.1 Architecture assessment method

- The major research claim here is that architecture assessment should be carried out as a **repeated, incremental activity**, considered as "normal" development practice, rather than a special, one-off (per project or product) action. This is the reason behind the extension to SAAM (where much reported experience referred to one-off, externally-driven assessments) as argued in section 4.4.2.

  The testimony and behaviour of stakeholders and architect support this approach. Especially when dealing with the fact that more requirements are generated by stakeholders than can be covered in a single assessment session (both cases), and that such assessments are really only interesting to stakeholders and architects if they represent a means towards a structured improvement in architecture practice.

- Industrial participants have a clear preference to **gradually introduce new concepts and processes into running projects**. Whether evolution or revolution is the (scientifically) proven best way to improve industrial practice is an open question, which is not addressed in this research. The assessments reported here have benefited from the willingness of people to leverage "familiar" concepts such as use-cases and native notations. A drawback of this approach, however, is that it may take several cycles before the full scope (the breadth aspect), or full power (the depth aspect) of the improved practices take hold.

  This should be expected and while it means that exercising the method was slightly restricted, it also means that those introducing the method should account for a gradual, customised introduction. They should make participants aware of this and establish

commitment to multiple assessment cycles as part of the acceptance and learning programme.

- The role of architecture assessments as **enabling stakeholders and architects to establish more clarity** (or the need for it!) in setting quality requirements as indicated in Kazman, *et al*., [1996] has been verified (again) by these exercises. Additionally this has been done where the participants have generated their own artefacts (using method techniques) and where external involvement was limited to a single method facilitator (this contrasts with external facilitation *and* architecture-creation skills reported as necessary in much assessment literature, e.g. [Clements and Northrop, 1996]).

  This demonstrates, as claimed in the critique of SAAM in section 4.4.2, that with the proper techniques, and of course motivated, capable stakeholders and architects, a company does not need *many* external resources to adopt architecture assessments. A small amount of external help is always helpful, especially for the neutral, politically naïve role of facilitation (see Chapter 5); and this level may change as experience increases. there is ample room for **expert consultants to teach and transfer the *process*, but the *content* should be owned and driven by the development team itself**.

- The high rate of **indirect change-cases** (see Appendix A) reported herein contrasts with those reported elsewhere in literature (e.g. [Bass, *et al.*, 1998], where typically 30%-50% of changes are already addressed by the architecture). This is explained because in this situation the development team itself is generating the requirements, rather than a customer who is remote from the system details. These stakeholders are familiar with what the system can and cannot do and therefore are more likely to know how to ask for changes that are not yet possible. This was especially so in the planning case, where the stakeholders were informed of the architecture's abilities immediately before generating their requirements.

6.5.2 Family stakeholders and qualities (interoperability and extensibility)

- Getting **explicit documented requirements and planning for the product families in the cases was difficult**. They did not exist prior to the assessment and where they were prepared during the assessment, the stakeholders did not have time to familiarise themselves with their implications.

  This can be explained by the fact that both cases were primarily in the technology driven pre-development phase (a normal approach to family orientation); and therefore the business artefacts were not complete, a situation which should change when moving to the market-release phase. The focus and emphasis on families raised by the assessment, and the awareness that the relevant artefacts (family-feature map, migration-map), asking reasonable questions about the family, raised a commitment from the respective business

managers to address these issues. In a sense the assessment raised the importance of these aspects and has triggered the responsible stakeholder to provide these artefacts to the architect. This suggests evidence of the need for repeated, incremental assessments. This is further proof (as claimed in Chapter 2) of the lack of attention given to strategic-family issues in conventional development methods; and the fact that the research deliverables are addressing obvious gaps in current practice. Unfortunately at this stage there is not enough evidence to comment in detail on the success of the techniques.

- In both cases, these family stakeholders had **not previously been involved** in such an interactive, requirements-driven discussion with the development architect and they responded very positively to the entire exercise.

- In contrast to the role of stakeholders as reported in SAAM (see Chapter 4), where they can act as knowledge-sources to support the external experts, in the FAAM method reported here the **stakeholders had organised and *documented* their own requirements** very much focused on the key family concerns of interoperability and extensibility. They had explored in their change-cases the areas indicated in this research (see Chapter 4) as central to information system families e.g. legacy-integration (for extensibility) and third-party systems (for interoperability). Interoperability and extensibility from a family context were solidly addressed in the cases, justifying the need for (as indicated in Chapters 2 and 4), and the design of (provided in Chapter 5) the guidelines and techniques derived in the research.

6.5.3 Method techniques

A summary of technique usage and stakeholder evaluation is included in Table 6.2 below.

- In general the techniques were positively received by the stakeholders and performed as expected (participants used them throughout). However, the application of the strategic family (business) oriented techniques demanded some effort and their completeness (as specified in Chapter 5) was compromised in the interests of progressing the assessments. This was largely due to the difficulty in establishing the documented knowledge (as indicated at the start of this section) about the family. The reasons underlying this are:
  - generic – the lack of familiarity of family issues in the wider information-systems development community;
  - specific – the fact that in the planning-case many of the stakeholders were not yet familiar with the family concept emerging from the pre-development project.

  However, stakeholders were (and did, as evidenced by the change-cases) encouraged to think families during their requirements-specification activities, and these issues dominated the assessment discussion. Repeat cases (as advocated by the incremental repetitive approach to architecture assessment underlying the method) are needed to thoroughly validate these particular techniques.

- The ***change case template*** was customised slightly in one case (medical), but regarded as very useful by the business-manager stakeholder in helping him think of extensibility issues. However in the other case the feeling was more mixed, details are reported in the appropriate section of Appendix A. The evidence here is inconclusive, the lack of consensus is most likely explained by the short-cut introduction in the planning case. However, the universal, open support for the idea of substantial (1 page instead of one line) stakeholder input, controlled by the stakeholders was explicitly found in both cases. This validates a basic tenet of the research – **the more refined the requirements, the higher the quality of the architecture response**.

- The time-scarcity in the planning case also led to some contradictory findings with respect to the term change-cases and the ***change-case-ranking-criteria***. This goes to prove that **method introduction and implementation cannot be short cut**, it takes time. Multiple repeat studies are needed to experiment with timing, but those advised in Chapter 5 should be adhered to if possible.

Table 6.2: Overview of technique usage and user evaluation in cases

Techniques\Method-steps	Medical (validation)	Planning (discovery)	Result
1. Family-Feature-map	++	+	Useful, but more evidence needed
2. Family-context-diagram	+	++	useful
3. Change-case-template	++	+/-	Useful, but more evidence needed
4. Migration-map	+	+	Useful, but more evidence needed
5. Change-case-guidelines	++	+	useful
6. Requirements-ranking-criteria	++	+/-	useful
7. 4+1 (SAAM)	++		Used in one case, demand for standard notation in other
8. Assessment criteria	+/-	+	Useful, but more evidence needed
9. Req-arch interaction metric (SAAM)	++	++	useful
10. Sensitivity metric (SAAM)	++	++	useful
11. Conformance statement	++	++	useful

Legend: (+)+ = (very)strong; (-)- = (very)weak; +/- = neutral

- Regarding **architecture representation**, (see Information Box 6-B) in both the interoperability and extensibility quality assessments Kruchten's [1995] *logical view* was used to describe behavioural aspects, this is especially appropriate for communication with stakeholders because of the problem-domain focus of this view. For extensibility (medical case) the *development view* was used to describe the effects of changes on source

code, this is again a static, build-time concern. When dynamic aspects of interoperability (planning case) arose, the logical view was embellished with some messaging logic, approximating the UML sequence-chart.

- An enhancement of the method provided by the cases was the fact that the ***review/refine artefacts*** did not need to be a group meeting; bilateral communication (with the facilitator informed) between stakeholders and architect was sufficient in both cases to ensure adequate input. This reflects a central point of the research that the **outline method** provided is a guiding framework which can, and should be, **customised** in recognition of the individual context of each assessment activity [Punter, 2001]. However, the importance of this check – group or bilateral – must be stressed to ensure that the whole group are fully prepared for the assessment meeting, especially of for any reason some of the participants cannot attend and substitutes are required (e.g. the planning case).

- While the assess architecture conformance meeting is focused on exploring how to resolve the requirements through impacting the existing architecture; the discussion typically also highlights ways to **address the requirements through non-architectural solutions** such as: implementation-configurations, modifying other systems. This prompting of the participants to examine the avoidance of problems, or re-allocation of change requirements outside the system family, can in the long run be of significant strategic value to the organisation.

- The role of the architect in providing technical detail to the ***conformance-statement*** is very important, and the facilitator and architect must work closely here, especially as it may be contentious. The power of documented feedback and rationale as provided by the conformance statement was especially manifest in the medical case. During the review of the first draft prepared by the facilitator, the architect re-examined his original impact analysis (from the assessment meeting) and came to a simpler (and in his words "better") solution to addressing the change-cases. Such **new architectural insights** obtained by the architect when considering changes is exactly the motivation behind, and justification for, the approach to architecture assessment herein.

Information Box 6-B: A note on architecture representation

The method does not enforce a particular architecture notation, for the simple reason that for the vast majority of organisations this would simply mean a barrier to architect-stakeholder communication under the guise of having to learn a "specific" notation.

As reported by those practising and studying architecture, there is (still) a plethora of notations in use. Most are of the simple, home-grown and multi-interpretable "box 'n line" style [Bass, *et al*., 1998]; while other more established (as regards semantics and penetration) notations dominate in particular industries and problem domains (see [Shaw and Garlan, 1996] for a thorough survey). Architectures belong to organisations, and many have adopted their own way of describing them, based on chance or reason.

While a standardised approach to describing software design artefacts is desirable (in fact it was requested by a stakeholder in one of the cases); it is not the goal of this research, nor should it be a prerequisite for success. The method advocated here is intended to be part of a larger development activity, with all its associated processes, communication-formats and tooling. Considering the current state of standardisation, it would be premature and restrictive on method acceptability to burden it with a notational constraint.

Architecture notation is not, however, treated as unimportant in the method. The increasing popularity of UML, and its support by case-tools, is making inroads to reducing the diversity in notations. The FAAM method recognises the importance of standardised notation and the emergence of UML by advising that, where there is the need or interest to (re-)create an architecture description for these assessments (a situation commonly encountered by the assessment pioneers, [Kazman, *et al*., 1996]), then the 4+1 views advocated by [Kruchten, 1995] using UML notation, is the preferred approach.

This approach leverages the real power behind the 4+1 (and UML), not the fact that there is a single notation; but more that the realisation that there are multiple views, each addressing different concerns (abstractions), needed to describe architecture. By the method and the facilitator advising the 4+1, it gives the signal to the architect that multiple views are important. Even if the current dominant view is not described in UML, the opportunity to separate or provide additional views (maybe using UML for this new view) is generally much more beneficial, and allows the adoption of standard notations to occur in an evolutionary way.

6.5.4 Implementation guidelines

Method introduction and facilitation is very much a people-centric exercise, and requires the full co-operation of the assessment sponsor and the architect. The following implementation lessons have been learned during the exercises.

- Including a **real customer** (a different type of external resource) in the assessment provided both credibility and quality input in the planning case; and may be considered as a good way to lower any internal resistance.

- The **small-steps-first approach** to method introduction has been indicated previously as appealing to stakeholders. Again the assessment-sponsor will be instrumental here in deciding just what aspects will establish buy-in and provide reward for the people and business respectively.

- Allowing **adequate time** for method implementation is very important. A solid introduction to the method with the participants as a group is especially important and showed to have a determining influence on how the techniques are applied. This has already been alluded to in section 6.5.3, more details are provided in Appendix A.

- Keeping the **meetings moving** is important and an experienced (in the method) third party (i.e. politically ignorant) facilitator is instrumental to this. In these cases shadow facilitators (internal to the company but eternal to the family-development team) were deployed to record detailed minutes, and to use company-internal knowledge where needed to close discussions.

- When there are a large number of participants, it may be difficult to **organise continuity** in meeting attendance; this should be striven for, because changing attendees often give rise to actual (or perceived, which can be equally damaging) communication loss.

- Large participant groups also mean that people may lose requirements in the selection cut. In organisational cultures where this may pose a threat to stakeholder morale a mitigation strategy is to (in prior consultation with the architect) try to establish co-authored, compromise change-cases, so that **all stakeholders are at least responsible (in some way) for a change-case**.

- In section 2.5 the **natural reluctance** on the part of architects to their work being "assessed" was observed as a natural phenomenon and a real issue in such assessments. Addressing this issue was claimed as a secondary benefit of the assessment method, where method experience would generate some practical knowledge on the issue. The important approaches pursued in these cases to prevent (or dispel) this reluctance were:
  - Portraying the assessment as a means of analysing *requirements* – thus shifting the focus of attention to the requirements rather than the architecture and emphasising the architecture's role as an important enabler of this analysis.
  - In the other case the *process*-aspect aim of the exercise was heavily pitched – i.e. architecture assessment not as a single "architecture exam" – but a means to institutionalise regular stakeholder-oriented architecture-reviews and thus improve overall architecture practice. This had the support of the architect, who then became a process-owner and was active in organising the review sessions.

  Deciding whether this re-emphasis is necessary and which of these strategies (or other strategies) is most appropriate depends on the local situation. The assessment-sponsor and facilitator should consider this in their pre-assessment meetings.

6.6 Summary

The research approach and deliverables, whose industrial validation is described herein, claim to contribute to the state of the art in architecture assessment in two main areas:

1. Extending the mainstream method (SAAM) to address *product-family stakeholders* and qualities (i.e. *interoperability* and *extensibility*); thereby enabling the assessments to facilitate a more inclusive stakeholder-architect discourse on family definition.
2. Providing *practical "how-to" techniques* and *implementation guidelines* to enable the family-*development team* to acquire ownership of architecture-assessment practice, and

instil its *repeated application* as part of normal family development

Both these claims have been investigated and the experiences reported in this chapter. Further, it is contended that the evidence and analysis presented provide sufficient proof to justify that the research deliverables have indeed successfully addressed the claims. The following set of concluding remarks recaps the evidence, and summarises the argument.

Of course, the work is not complete and the experiences gained have provided motivation for further investigations of the method design in future cases studies, and indeed insights to enhance the design. These issues will be considered in the next chapter, which provides an overview and conclusion to this research.

6.6.1 Case studies – concluding remarks

With respect to the primary research aims repeated here in section 6.1:

1. **Active involvement of family stakeholders** – the structured inclusion of those strategic-level family stakeholders excluded from conventional development processes and tools has been prototyped. It has been shown to be effective in uncovering "first-hand" requirements detail and establishing buy-in for assessment practice and the subsequent architecture findings. It was also regarded as a rewarding experience for both stakeholders and architect.
2. **Focus on the family-relevant system qualities of interoperability and extensibility** – the cases have exercised the ideas, techniques and architecture views associated with analysing interoperability and extensibility from a family context. This has demonstrated the successful application and extension of the general SAAM architecture assessment method to the area of product families (the FAAM), addressing one of the main aims of the research.
3. **How-to techniques for self-assessment** – the cases have supported the central notion of the research; that the provision of simple techniques to enable stakeholders to generate and present their requirements (in relation to family interoperability and extensibility) is both possible and preferred. These techniques are very important in making architecture assessments amenable to take-up by the broader industrial community; and in ensuring the concept of assessing architectures becomes embedded in the "normal" development process. Evidence (via testimony from industrial practitioners) that such an approach is desired has been provided.

Additional important findings from the case studies include:

- The cases described here have provided repeated evidence (empirical and testimonial) of the claim (e.g. [Kazman, *et al*., 1996]) that architecture assessment is a valuable contribution to **increasing architectural understanding amongst stakeholders**.

- They have further provided preliminary evidence that **organisations can themselves take more direct ownership of these assessments** and establish commitment and discipline to incorporating these reviews as part of their normal development process.
- Evidence has been established that **stakeholders and architects can provide the required artefacts** when given some techniques to do so. This contrasts with much other reported architecture-assessment research (see [Kazman, *et al*., 1996], [Bass, *et al*., 1998]) where often persons external to the team must invest significant effort in actually preparing the necessary requirements and architectural artefacts.
- The **extension of the conventional use-case description and templates to capture the system-quality requirements** for interoperability and extensibility has been exercised. This is a natural evolution as both these qualities are closely related to functionality, but it indicates that organisations can leverage existing experience in adopting a structural approach to these (and other), often ignored, system qualities.
- The primary architectural views (in terms of [Kruchten, 1995]) used to address interoperability during the assessment is the *logical view*. Extensibility was assessed using the *logical*, and *development views*. This is a valuable contribution as guidance in determining which architectural views are appropriate for assessment of which stakeholder concerns. Of course it should be noted that *all* 4 views (and others!) are required to fully describe the complete set of architectural aspects necessary for the development (as opposed to stakeholder) analysis of these qualities.

Chapter 7
Reflection – evaluation and conclusions

7.1 Purpose

The purpose of this chapter is to reflect on the practical experiences (Chapter 6) of the research products (Chapter 5) with the aim of presenting and validating the research contribution to:

- the body of associated scientific knowledge;
- the state of the art of industrial practice.

The learning aspect of the practical experiences will also be leveraged through incorporating the lessons learned as evidence supporting the research claims. Bearing in mind the limited focus of the research area and the broader range of opportunities for improving the problem domain, some suggestions for further research are provided.

7.2 Structure

Firstly the general conclusions of the research will be provided. Following an examination of the scientific value of the research results, these will be compared against the research aims and questions presented in Chapter 2. This comparison will present, per research question, the relevant contributions of the research to the area of information-system family architecture assessment. The contributions will be distilled from the analysis, design and implementation phases of the research, reported in Chapters 4, 5, and 6 respectively. In closing, some recommendations for future research are made.

7.3 Conclusions and verification against requirements

This section presents the findings of the research as inferred from the various design-steps reported throughout this work. These findings represent the contribution of the research to the scientific and practical body of knowledge. The theme here is general; those interested in actually implementing architecture assessment according to the ideas presented in this work, are referred to the overall conclusions of Chapter 6 which provide more detailed observations and method enhancements, including a dedicated section to the implementation lessons learned.

7.3.1 General conclusions

The central motivating theme of this research is the fact that conventional development approaches to information-system families have provided insufficient support for:

- strategic product-family stakeholders;

- their associated system-quality requirements;
- the importance of architecture for the family business.

As a means to address this problem, the idea of stakeholder-oriented architecture assessment – with its emphasis on promoting a stakeholder-architect dialogue on requirements and their architectural consequences – was seen as offering a promising solution approach. This approach (based on SAAM [Kazman, *et al.*, 1996]) has been followed, and important additions to established architecture-assessment practice provided to increase the speed, ease and effectiveness of its application in the industrial community. The most important contributions of the research have been its:

1. explicit attention to, and active inclusion of, family stakeholders in architecture assessment;
2. addressing the specification and assessment of the important product-family qualities of interoperability and extensibility;
3. provision of a method with practical "how-to" techniques, to enable those developing families to apply the assessment method themselves and embedding it as a normal development practice. Such general purpose techniques help make SAAM a more repeatable process thereby increasing its (process) maturity and industrial penetration.

7.3.2 Scientific value

Proving the scientific nature, let alone the scientific value, of design-oriented research into social systems using case studies is not a trivial task (see e.g. [Yin, 1994]) and has had an eventful past (perhaps present and future). Chapters 2 (research design) and 6 (case study design) contain discussions on the relevance-of, and rigour imposed-on the case-study method in this research. This section will summarise those aspects of the case-study experiences that bear on the scientific value of the conclusions drawn from them.

Scientific value is generally measured by questioning the validity, completeness, and transferability of the research object. In terms of the collected process and techniques developed in this research, the fact that the aim of the research is to provide *practical* support to *practising* professionals, means that the practical value of the research results will weigh heavily on the evaluation (see ['t Hart, 1997], Chapter 3 for a similar discussion).

Establishing the *validity* of the method and techniques would be best addressed by asking the question "does the application of the method improve the quality of the architecture?" This seems a reasonable standpoint, but there are some significant difficulties considering the "reality" of the context in which the method is applied. Firstly, establishing the quality criteria of the architecture under consideration is part of the method's role, and therefore not an independent measure (in the traditional scientific sense) of the method's effectiveness. Secondly, establishing exactly what the quality of architecture is, both before and after architecture assessment, is a non-trivial exercise both theoretically and practically seeing the abstract and long-term nature of architecture. The requisite resources of both theory and

practice were outside the scope of this research. Thirdly, even if architectural success, or failure, could be measured; it could be caused by many other influences outside the control of the method presented here.

The practical impossibility of answering (in a scientifically-absolute sense) such a validity question means that the *analytical generalisability* approach towards interpreting case-study results [Yin, 1994], the reviewing-abilities (sanity-checking) of domain experts and the testimony of participants are the pragmatic approaches towards determining the validity of the findings. A further, very important, point is that the method claims to support stakeholders and architects in *their* task to establish architecture quality; architecture quality is their responsibility, and depends on their work. For this reason, the positive reaction regarding the role of the method in motivating and enabling the assessment meetings of two different product families, across two organisations, with different architectural approaches and scope is compelling evidence.

Method *completeness* ("are all aspects of interoperability and extensibility in product- family architecture addressed?") runs into the same arguments regarding constraints as before. Completeness in an immature domain, with multiple contextual influences is unlikely. The method is, furthermore, a first step, repeated application may expose weaknesses, which can through learning be used to increase the coverage of the method. The completeness of the method will therefore improve with exposure, but it can never be complete.

While the number of case studies was small the *transferability* of the method is estimated as high because of the fact that only the facilitator needs to be expert in the actual method (the participants prepare all other artefacts and the cases showed this to have been reasonably well done). There were 12 participants used across both cases from a wide variety of backgrounds (8 stakeholders in the planning case), so the scope of stakeholders used has been high. A non-method expert provided method introduction in one of the cases, and while there were some communication problems these had more to do with logistics rather than inherent aspects of the method. The implementation lessons learned during the cases, which focus on how it can be practically implemented, are provided in Chapter 6 and will increase the transferability.

Despite the constraints inherent in case-study research, the practical arguments above support the scientific underpinnings of the research results. There is sufficient research evidence to make some *tentative* generalised conclusions regarding architecture assessment in information-system families. Providing *final* generalised conclusions warrants additional research outside the scope of this project. In the spirit of self help promoted throughout this work, Kennedy's [1979] advice – that the general applicability of design-specific knowledge must be determined by those persons who want to apply the results to other design cases – is suggested.

7.3.3 Contributions versus requirements

The following research questions have been formulated in Chapter 2 as necessary to satisfy the aims of the research. These questions and the answers provided are summarised here. Major research findings are indicated in bold text after relevant questions.

1. Who are the stakeholders of information-system families, and how do we elicit architectural requirements for assessment from them?

❑ **Product-family stakeholders have familiar faces, but are wearing new hats**

In Chapter 3 it was shown that families are not "greenfield" in nature, they require the presence of established market and technology combinations and generally grow organically from situations of:

- multiple, related, but individually developed products (the family suite approach);
- a single (large) application set that is customised to customer wishes (the system variant approach)

This means that family stakeholders will evolve largely from existing product stakeholders and will adopt their stakes to reflect the additional family concerns. An example of this phenomenon from the development stake is the fact that in the pre-family situation there was development, whereas in the family context development may have been divided into generic-asset development and (end-)product development (see [Clements and Northrop, 1999]). The existing development stake(holder) has gained some extra concerns, balancing between platform issues and application issues. More examples of these extra family concerns and their respective stakeholders have been provided in Chapter 4.

❑ **Family development is characterised by a strategic (customer) rather than operational (user) stakeholder focus**

A very important aspect of family development is its strategic long-term nature; the family architecture framework (see Figure 7.1) generated in Chapter 4 highlights this aspect, and its absence from most conventional development practice. The framework was used to identify these strategic stakeholders (especially the business management and increasingly important customer) and their stakes. In particular, it was illustrated that the customer was the dominant external stakeholder (in contrast with the exclusive attention awarded to the user in conventional development) and had stakes in the financial, support and development aspects of the family.

In order to reflect the fact that effective family management must be confined to a small team, and the fact that individuals can fulfil multiple roles; the following set of *primary* family stakeholders have been identified:

- business management;
- product (line) management;
- customer/operations support management;

- development management.

Furthermore, guidelines towards what other stakes these stakeholders can represent and the associated family concerns are also provided.

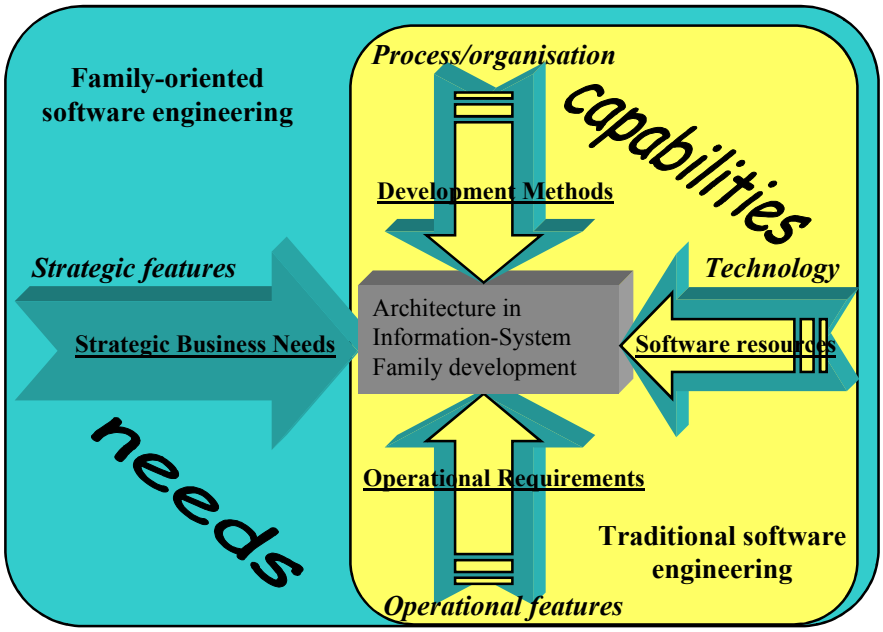


Figure 7.1: Strategic stakeholder focus of family architecture

❑ **Conventional requirements techniques can be extended (Use-cases ➔ change-cases) to express quality requirements**

With respect to extracting requirements for the system qualities, the increasingly accepted method of stakeholder oriented requirements engineering based on use-cases [Jacobson, *et al.* 1992], was seen as a means to leverage existing best practice from conventional development. This follows the approach advocated by the most established architecture assessment method (SAAM, ATAM [Kazman, *et al.*, 1998]).

System qualities deal with system-wide properties, underlying, but not directly associated with usage (or more correctly, a particular user). Such non-user aspects are of most interest to the strategic stakeholders, who are concerned with the longer-term issues of how the system fits with the expected customer- and producer-environment changes. The change focus of the system qualities e.g. maintainability (changing parts to prolong life), scalability (dealing with changes in workload), portability (changing the underlying computing resources), has meant that system qualities have tended to be ignored in mainstream development due to the difficulty of making these qualities concrete ("you can't predict change"). Just as use-cases describe representative scenarios of desired actor-system usage; the term *change-case* [Bennett, 1997] is adopted here and used to elicit representative use-cases of the types of changes that the system must support. This is a means of providing illustrations of the most important expected changes so that there is something explicit to carry the quality into development.

- **"How-to" techniques are needed to support stakeholders in generating quality requirements**

In order to support these (previously ignored) strategic stakeholders in preparing explicit requirements for the system qualities, guidelines in the form of a question checklist and simple graphical representation of the system and its quality-context have been developed. This extends current "what-to" practice by providing stakeholders with "how-to" tools so that they are empowered to implement the method. More details of the techniques are provided below.

- **Interoperability and extensibility are key qualities for product families. Both the *intra*-family and (conventional) *extra*-family aspects of these qualities must be supported.**

Flexibility (the ability of the system to accommodate change) and reuse are the key elements of product families. In this research two types of flexibility were addressed in order to operationalise the assessment method:

- interoperability – "*The ability of two or more systems or components to exchange information and to use the information that has been exchanged"* (IEEE std. 610.12-1990) *...in order to support a set of coherent business processes which require the co-operation of the systems or components*".
- extensibility – *The ability of the system to accommodate the addition of **new** functionality*

These are key flexibilities in terms of product families. Information systems are typically operating:

- in supporting *parts* of an overall business process with significant human involvement;
- as part of *multi-party* value chains both within and between separate enterprises.

This means that all information systems, whether developed as part of a family or not, must provide facilities so that they can be *extended* to address additional (as yet unsupported) functionality for the business process or user. Similarly the must be able to *interoperate* with external, third-party systems in supporting multi-system processes. This conventional, external-oriented, approach to interoperability and extensibility is termed *extra*-family in this research.

When a producer competes in a family model however, interoperability and extensibility become a matter of internal concern (see Figure 7.2):

- Customers expect to buy entry-level ("low-range") systems, and later (depending on financial or maturity circumstances) migrate to a higher-class system from the family. In such cases the uplift of the low-end system must proceed seamlessly and preferably cheaply (i.e. reusing data, information structures, and where possible other software and hardware infrastructure-elements). In this case the low-end variant is being *extended* to the functionality of the mid-range variant.
- Producers realise the economies of families through reusing common assets across multiple application systems, both within and between generations. This means that such

asset components must interoperate with a range of other application components, and also this must be possible as both assets and applications proceed through independent upgrades (extensions) across time. Furthermore, in families comprising a suite of related products, the family members may have a higher level of integration with each other than with external systems as a means of discriminating the family in the market.

Therefore, where previously these "ilities" could be seen as extra-features and perhaps provided a source of explicit revenue, in a family context they become central to family viability, and are expected to be part of the standard services offered. This new internally driven demand for increased levels of interoperability and extensibility is termed *intra*-family in this research.

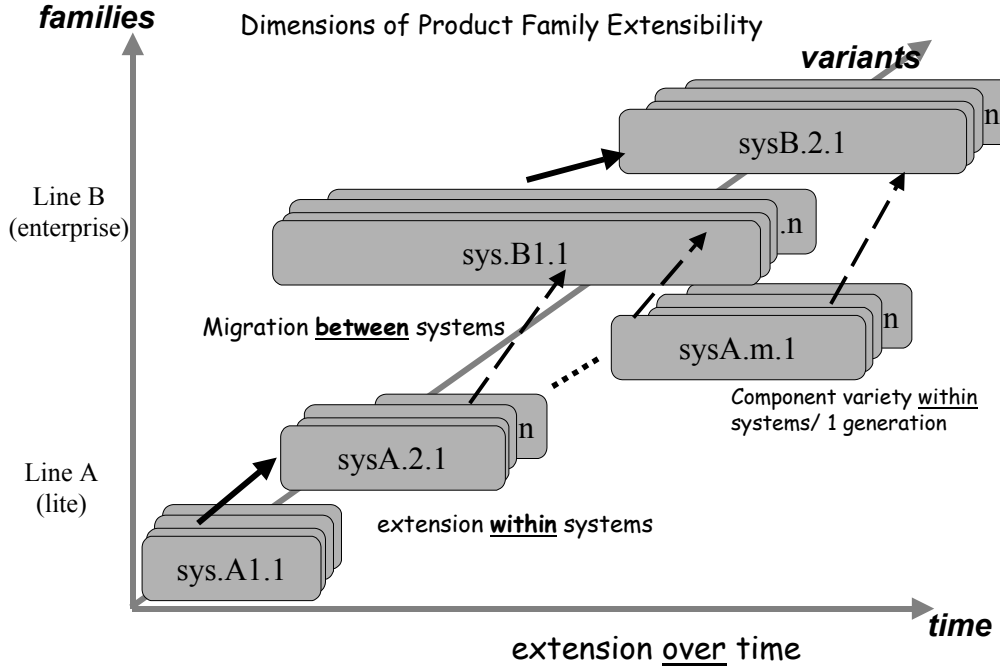


Figure 7.2: extensibility in a family context

- **Representation of information-system families by strategic stakeholders is an immature discipline**

In the case studies implemented in this research, and through other contacts, it seems that the explicit modelling (description) of the scope, content and planning of product families in the information-systems domain is not well established during initial family definition[1]. Although some efforts towards supporting stakeholder modelling of product families appear in literature, most have a dominantly functional, time-independent approach e.g. [Erens, 1996]; class diagrams with inheritance and variation points as advocated by Jacobson, *et al*. [1997]. A more feature-based approach amenable to the strategic stakeholders and the early (pre-functional-requirements modelling) stages of family definition is the spreadsheet-like PULSE method devised by the Fraunhofer institute [DeBaud and Schmid, 1999]. This addresses the

[1] Bass, *et al*. [1998, pp. 370] have reported similar informal beginnings to the common and variable aspects of family domain definition in their CelsiusTech case study.

business-definition aspects of family development and includes as one of its elements a product map to provide an overview of the features important to the family both now and in the future. This is an extensive method, which is undergoing active development, and has some accompanying tool support, but it is not yet well established in a commercial sense.

It was not the aim of this research to derive or implement a complete product-family description tool. The assessment activities would have undoubtedly benefited from such a description, and indeed some simple techniques were developed (see next section) in order to encourage stakeholders to capture a family context for the qualities of interoperability and extensibility. In the cases explored, families were relatively new concepts to the organisations and were still something more intuitive and vague than explicit and clear to the organisation as a whole. While no universal claims as to the state of explicit family representation can be made based on the two cases explored, it seems reasonable to postulate that there is still work to be done in providing structural support for family modelling to those initiating family development.

- **Simple tools to orient stakeholder thinking towards family issues, coupled with a structured format for expressing change-cases provide an effective mechanism to support stakeholders in generating and representing family-oriented system-quality requirements. This is especially so when organisations are in the early phases of developing product families.**

Defining family system-qualities is a multidiscipline, and therefore multi-stakeholder, negotiation process. These stakeholders are typically unaccustomed to dealing explicitly with system qualities, or with families. Simple, rather then complex, tools are most appropriate in building up stakeholder confidence and skill in this activity. This research has provided, and successfully tested, some such tools to facilitate stakeholders in generating, and describing family-oriented requirements for interoperability and extensibility. These are:

- ***Family-feature-map*** – a tabular summary (similar to the PULSE product-map, [DeBaud and Schmid, 1999]) of the features of the product family; their distribution across family members, and an indication of their variability.
- ***Family-migration-map*** – a graphic illustrating the various members of the proposed family and their ancestors and planned-descendants over time. This tool captures the past/future aspects of family indicated as a challenge to family development – analogous to a family tree in genealogy.
- ***Family-context-diagram*** –a graphical overview of the various systems/components which are (or can-be) present with the family in its operating environment.
- ***Change-case-guidelines*** – a set of questions used to encourage stakeholders to generate change-cases for a specific system quality.
- ***Change-case-template*** – a structured means to describe the quality requirements, facilitating explicit elaboration of the rationale, the desired situation, and particular quality-related- and family-context details.

These tools have been applied and validated in the industrial case studies reported in Chapter 6. They have been shown to help stakeholders in generating and formulating their quality requirements and have been subsequently used by architects in assessing the architectural impact of the requirements. It should be noted that the family-feature-, and migration-map were not extensively tested in the case studies due to the absence of a high-level business-roadmap for the family, or the absence of the time to construct such a plan. Proving the validity of these tools, therefore, depends on logical reasoning and opinion rather than hard implementation experience.

- **The following architecture views are useful when assessing:**
  - **Interoperability – logical and process view**
  - **Extensibility – logical and development views**

  in early-phase family-architecture definition.

Architecture representation is not universally standardised apart from the ubiquitous "box-'n-line" style. Specific architecture description language (ADL) developments are still largely confined to the research domain, with some few commercial exceptions (e.g. ROOM); and most are still too immature for industry, especially in their support for product-family architectures [Bass, *et al*., 1998]. For this reason (see also Information Box 6-B) the assessment method proposed here does not *mandate* any single architecture representation. Instead the 4+1 views described by [Kruchten, 1995] (see Figure 7.3), and supporting notation in UML, is *advised* as a comprehensive means to represent architecture.

The importance of architecture views is recognised in this research, and one of the contributions is to provide guidance on which views are relevant for the particular system qualities under consideration. The case study implementations illustrated that:

- The *logical view* (comprising the static relationships between important application-domain entities) was used in communicating architectural issues associated with both *interoperability and extensibility*. This is because the logical view, with its domain bias, is most familiar to stakeholders, and contains terms they use to express their requirements. Further the logical view is usually the first view prepared in the architecture life cycle and is used to derive the process and development views.
- The *process view* comprising the dynamic and synchronisation relationships between individually managed units of execution) was also used to explore the dynamic aspects of *interoperability*, if stakeholders raised these issues. The reason here is that interoperability is (by definition) associated with systems or components communicating to support activities, and is commonly associated with functionality where the actor is simply another system rather than a human user. The dynamic aspects of system behaviour modelled by the process view is, therefore, very appropriate for interoperability change-cases.
- The *development view* (comprising the software modules in the development environment) was used with the logical view in explaining the impact of *extensibility* requirements. This can be explained by the fact that extensibility refers to adding new

functionality to the system, which typically means adding or modifying source-code in the software modules modelled in the development view. Assessing the impact of new functions will naturally concentrate on what key domain entities (the logical view) and what key build-time development items it affects.

Two very important observations here are:

- no statement is (or can be) made on the granularity of these views, this depends on the nature and detail of the requirement [Kazman, *et al*., 1996];
- no claim is made on the sufficiency of these views, research experience as reported in Chapter 6 has shown these to be useful (even necessary), other views may also be used depending on the details of the requirements and other available architecture views.

❑ **The logical and development views are closely associated with addressing family concerns**

Families differ most from single-product development in the attention given to strategic, long-term issues. Strategic stakeholders (e.g. business-manager and customer-support manager) are concerned with future market developments to determine e.g.:

- product mix – to address the market variety;
- platform content – which must be reused (to be repaid) across multiple product lines and generations;
- upgrade paths – to accommodate future expansion in scope.

This is why the non-operational system qualities have a higher profile in family development. Bennett [1997] distinguishes between the run-time (when the system is operating in its usage environment), and build-time (when the system is in development or maintenance) concerns of a system's life cycle. The build-time issues are of most concern to family stakeholders as the key system qualities underlying ongoing family development (e.g. configurability, modularity, extensibility, portability) are first addressed in these high-level, fundamental views of the application and technology domains. Figure 7.3 illustrates how the more operation-, and technology-specific decisions are incorporated into the architecture from the basic logical view. Empirical evidence for the importance of these views for strategic issues is provided in the medical case study on extensibility reported in Chapter 6 (interoperability is more closely associated with run-time aspects).

It must be noted, as stated in Chapter 6, there are no short cuts; *all* views are needed to *develop* families. The importance of the views highlighted above must be seen in the context of early life-cycle *stakeholder* assessment of family-system qualities.

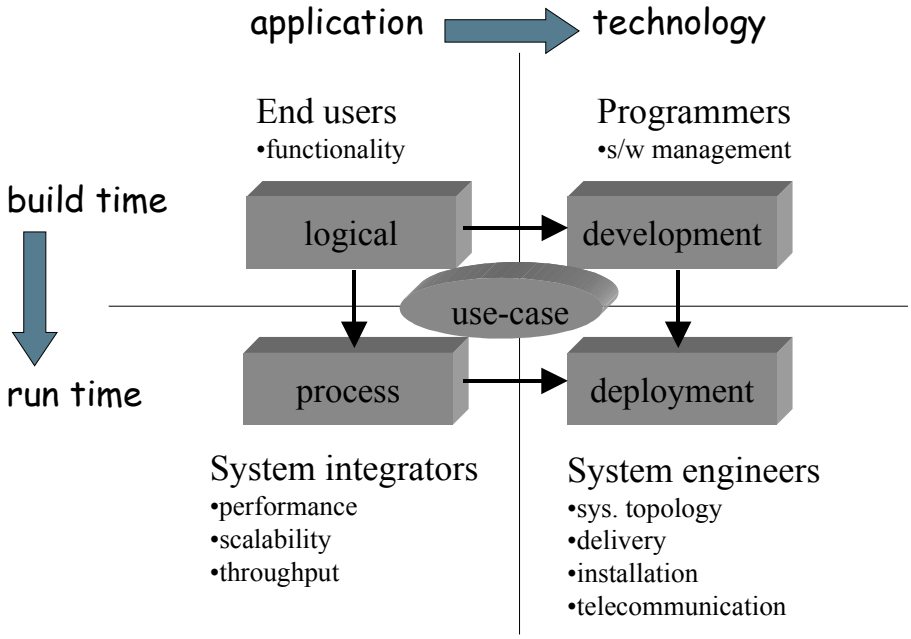


Figure 7.3: 4+1 architecture views – their life cycle and domain context (adapted from [Kruchten, 1995], repeated from Chapter 4)

2. **How can the architecture be assessed with respect to the system-quality requirements?**

- **The Family Architecture Assessment Method (FAAM) provides a step-by-step process for stakeholder-centric, requirements-based assessment of family architecture.**

The FAAM method (as described in Chapter 5, and illustrated in Figure 7.4) with its associated techniques and implementation guidelines, and its solid basis on the established SAAM method has been shown to enable stakeholders and architects to engage in structured architectural discussions on important family requirements. Currently the techniques are limited to supporting stakeholders and architects in assessing the important system qualities of interoperability and extensibility.

The chief contribution of the practical techniques has been to provide an initial framework within which to develop the SAAM approach so that it becomes a *repeatable* (in CMM terms) process independent of external architecture experts. An important discriminating aspect of the method is placing responsibility for requirements and architecture description to the appropriate organisational stakeholders; and focusing the external facilitator on process stewardship tasks.

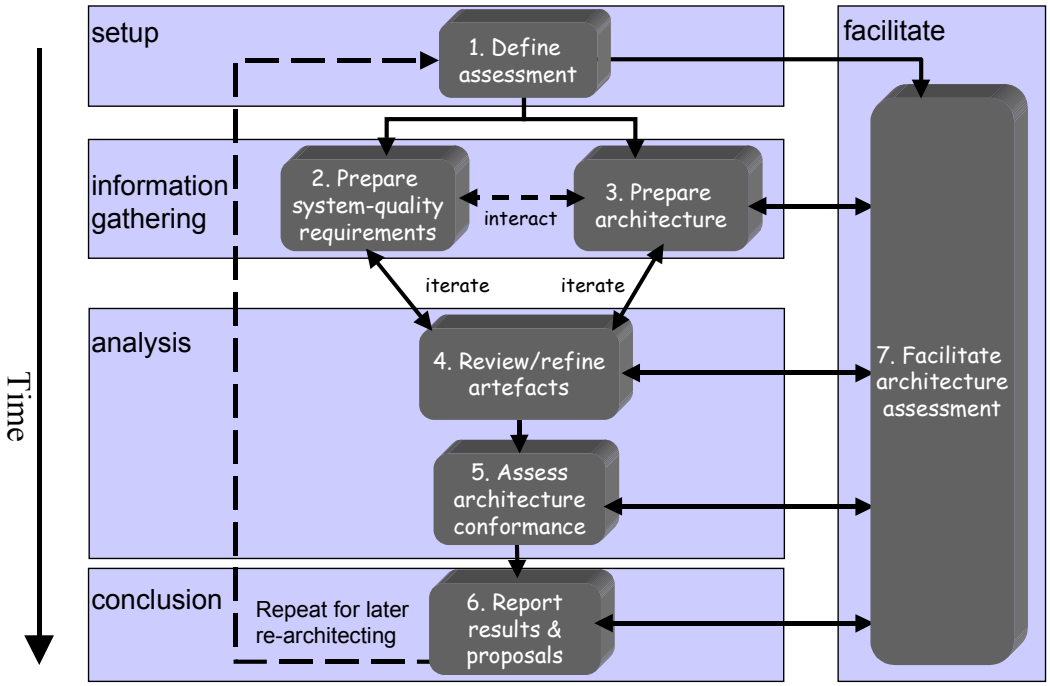


Figure 7.4: FAAM – overall process

- **The *self-assessment* approach underlying FAAM is necessary to embed the method as part of normal development practice, thus enabling architecture assessment to be a key competence in developing product families.**

Most reported architecture assessment [Bass, *et al.*, 1998] is done with the significant involvement of people external to the family-development group; this is normal when:

- the object of the exercise is a one-time (for the project) certification exercise to e.g. screening sub-contractors;
- an organisation is taking its first steps to learn and refine the architecture assessment technique.

The position adopted in this research, however, is that architecture assessment is an excellent mechanism for structuring ongoing stakeholder-architect dialogue as a means of (continuously) improving architecture development. This is analogous to the Fagan [1986] inspection approach in document/code-review. Like the inspections, and any key professional practice, it is something that must belong organically in the development group, it cannot deliver full, long-term benefits if it must be mandated from, and carried by, external drivers.

This approach has been validated by stakeholder reaction in the reported case studies (see Chapter 6); they universally expressed the opinion that:

- such assessments should be planned in all development projects;
- they expected feedback from the architect on the further design and implementation of their requirements;
- other qualities and requirements not addressed in the assessment-session or architecture-conformance meeting, respectively, should not be forgotten.

Architects were also interested in the method, not as a one-off exercise for the current project, but as a means to improve architecture practice in the long term. This indicates a demand from product-family developers for an incremental, repeated approach to architecture assessment within their organisations.

Further, if architecture assessment is to make a significant impact of the industrial community as a whole (scale up), then it must be amenable to implementation by a wider group of people.

- **Techniques enabling participants to implement the method are necessary for self-assessment to work. These techniques will be domain and quality specific, and should be developed through industrial exposure. The method framework facilitates the inclusion of such techniques and provides an initial set for the qualities of interoperability and extensibility in the information-systems domain.**

There is a reason for the lack of attention to "how" issues in most improvement-process research, that is the reality that the local context will exert a large influence on how the process is implemented (see [Punter, 2001]). This means that providing a *complete* cookbook on how to *precisely* implement the various process steps is impossible. It does not, however, mean that there is no need for *some guidance* on how to implement the process. The research has approached the problem of providing "how-to" guidance in two ways:

- Firstly, there is a framework (see Figure 7.5) which recognises that on top of the generic "what" of architecture assessment proposed by Kazman, *et al.* [1996] there is a place for domain-specific and quality-specific add-ons providing guidance on "how-to" implement the "what" process in specific domains, and for specific qualities.

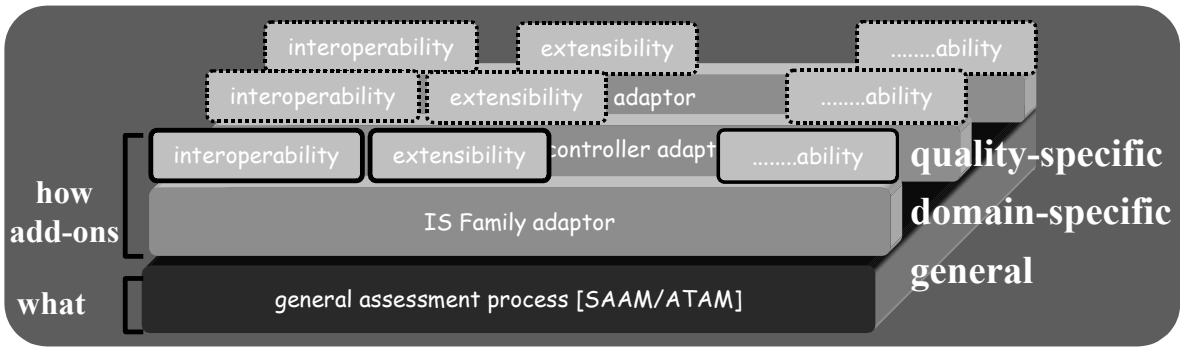


Figure 7.5: conceptual framework for adding domain- and quality-specific implementation guides for architecture assessment

This research has focused on one such area, information-system families, and two associated system qualities, interoperability and extensibility. The "how-to" techniques provided to aid completion of the method steps have been described, and have been validated through industrial exposure. The aim (see section 7.4 below) is that the domain and qualities addressed here, and other domains (e.g. embedded controllers) and qualities (e.g. configurability) are further explored by other researchers and practitioners, so that a wider range of implementation techniques will be available.

- Secondly, this research report, with its extensive focus-on, and description-of method design and implementation, provides documented experience of how research ideas are actually transferred to industrial practice. The plea from practising professionals for "fishing stories" [Fritz, 1997], – recounting both successes and failures in actually doing things in a practical setting – is made in response to a perceived failing of academic research to address the industrial application of ideas. Although such experiences are person-, organisation-, or application-specific; they provide insight which others may adapt or generalise to help their individual situations. Answering such a plea has been a strong motivation force underlying the research.

Chapter 6 presents evidence that the techniques provided have been usable, and useful-to family stakeholders in assessing their associated family architectures. All 12 participants, from two organisations have tested the techniques in an industrial context.

3. How can the results of the assessment be recorded and communicated back to the stakeholders?

❑ **The conformance-statement, which presents a structured report on the requirements, architectural impact, and proposed actions, is used to record and communicate the assessment results to the organisation.**

The method advocates the formal reporting of the assessment results, and provides a template (see Chapter 5) to structure such a report. It further advocates that the proposals from the report should be incorporated as part of the ongoing project deliverable and progress-tracking mechanisms. Technical content in the report is the responsibility of the architect, and the architect should take ownership of the findings and proposals in subsequent design activities.

7.4 Future research directions

While this research has fulfilled its requirements and made a useful contribution to the practice of architecture assessment in information-system family development, it is also true that throughout the course of this research many more questions arose than could be answered. Some of these avenues for further investigation are presented below:

1. *Carry out more testing with the family-related techniques* (family-feature-map and migration-map). With respect to the method and techniques presented here; it is fruitful to increase the data available for their validation and to refine them through experience. In investigating the incremental approach of the method it is also useful to plan follow-up ongoing-assessment (i.e. repeating the architecture conformance meeting) for the same requirements set to judge follow-up and discover what views, insights arise as architectural evolution progresses.
2. *Adding similar family-context, and implementation-support techniques to the decision-support metrics and steps provided by ATAM*. With respect to architecture assessment in general; the researchers at SEI have been busy since this research started and have already

expanded SAAM to the ATAM method (see e.g. [Kazman, *et al*., 1999]), which addresses the fact that architecture must trade-off between multiple (conflicting) system-quality requirements. The research reported here has taken SAAM as a basis and extended it so that it accommodates family requirements and has provided practical techniques and guidelines enabling participants to do-it-themselves. This is the "DIY" version of SAAM (single, independent quality analysis) and has not actively used the decision-techniques offered by ATAM (multi-attribute, trade-off analysis). Enriching FAAM with ATAM concepts will also help to increase the exploitation of ATAM, and is a natural extension to the approach reported in this research.

3. *Defining appropriate metrics to be used as part of ongoing assessment improvement*. Process metrics are necessary in order to monitor the costs and benefits of architecture-assessment, and especially to measure assessment performance to establish operating norms for planning (e.g. how long does it *usually* take to rank 20 change-cases with 7 stakeholders). This will help to increase the maturity (to e.g. "managed", level-4 in CMM terms) of architecture assessment process. Deriving relevant metrics (as distinct from the example provided) and the associated assumptions linking them to successful assessment and successful architectures is an open research question.
4. *Customising technique usage to the various types of families and assessment situations present in industry*. The techniques presented here are developed to support family-based assessments and have been show to be useful in the context of both *discovery* and *validation improvement-oriented* (see Chapter 4) assessments. The various techniques have not been related to the family typology referenced earlier in Chapter 3 of the research (suite, variant, component [Jacobson, *et al*., 1997]). The types of customisations and their motivating influences could reveal important information on the various family types.
5. *Supporting strategic family stakeholders in framing their family concerns*. Method experience revealed that the representation of families from the stakeholder perspective was difficult, and there were only a handful of approaches emerging from industry and academia. Research attention is needed in addressing the business-oriented modelling of families, in particular how can the stakeholders work co-operatively to communicate and integrate their views?
6. *Expanding the initial process and techniques of FAAM to a library of process components for other domains and qualities*. The method and conceptual assessment framework presented here are initial steps on the road to providing a comprehensive set of domain-, and quality-specific architecture assessment techniques. The long-term aim is that through increased industrial and academic exposure in these (interoperability and extensibility for IS families) and other domains and qualities; those domain-, quality-specific parts will mature to form a *library of process components* [Dolan, *et al*., 2000] providing a means to customise the general assessment method. This depends heavily on adoption and experience reporting by the practising community. As custom techniques are provided by others; new possibilities for combining domain extensions may emerge, and as the general understanding of the system qualities increases, it may be possible to derive domain-independent techniques.

References

[Abowd, *et al.*, 1997]

Abowd, G., L. Bass, P. Clements, R. Kazman, L. Northrop and A. Zaremski; *Recommended Best Industrial Practice for Software Architecture Evaluation*; CMU/SEI-96-TR-025; Jan. 13 1997.

[Van Aken, 1994a]

Aken, J.E. van; "Management Science as Design Science: the Regulative and Reflective cycle"; *Bedrijfskunde* (66:1) pp. 16-26; (Dutch) 1994.

[Van Aken, 1994b]

Aken, J.E. van; "Developing Scientific Knowledge for Management Professionals from a Players' Perspective: the Role of Design Models and Heuristics"; *Management & Organisatie*;(4) pp. 388-404; 1994.

[Argyris, *et al.*, 1985]

Argyris, C., R. Putnam and D. McLain Smith; *Action science – Concepts Methods and Skills for Research and Intervention*; San Francisco; Jossey-Bass; 1985.

[Bass, *et al.*, 1997]

Bass, L., P. Clements, S. Cohen, L Northrop, J. Withey.; *Product Line Practice Workshop Report*; Technical Report: CMU/SEI-97-TR-003 ESC-TR-97-003; Carnegie Mellon University; Pittsburg, PA 15213-3890; June 1997.

[Bass, *et al.*, 1998]

Bass, L., P. Clements and R. Kazman; *Software Architectures in Practice*; Reading MA; Addison Wesley; 1998; ISBN 0-201-19930-0.

[Bass, *et al.*, 1999]

Bass, L., G. Campbell, P. Clements, L. Northrop, D. Smith.; *Third product Line Practice Workshop Report*; Technical Report: CMU/SEI-99-TR-003 ESC-TR-99-003; Carnegie Mellon University; Pittsburg, PA 15213-3890; March 1999.

[Bengtsson and Bosch, 1999]

Bengtsson, PO., and J. Bosch; "Architecture level Prediction of Software Maintenance"; *Proceedings of International Conference on Software Engineering*; 1999.

[Bengtsson and Bosch, 2000]

Bengtsson, PO., and J. Bosch; "An Experiment on Creating Scenario Profiles for Software Change"; *Annals of Software Engineering*; February 2000.

[Bengtsson, *et al*., 2000]

Bengtsson, P. O., N. Lassing, J. Bosch, and H. van Vliet; *Analyzing software architectures for modifiability*; Technical Report HK-R-RES–00/11-SE; Högskolan Karlskrona/Ronneby; 2000.

[Bennett, 1997]

Bennett, D. W.; *Designing Hard Software: the essential tasks*; Greenwich CT; Manning Publications Co.; 1997; ISBN 1-884777-21-X.

[Berard, 1995]
Berard, E. V.; *Basic Object-Oriented Concepts*; Gaithersburg, Maryland; The Object Agency Inc.; 1995. URL: http://www.toa.com

[Bertrand and Wortmann, 1981]
Bertrand, J.W.M, and J.C. Wortmann; *Production Control and Information Systems for Component-Manufacturing shops*; Amsterdam; Elsevier; 1981.

[Booch, 1993]
Booch, G.; *Object-Oriented Design with Applications, Second Edition*; Redwood City, CA; Benjamins-Cummings Publishing Company; 1993.

[Bosch, 2000]
Bosch, J.; *Design and Use of Software Architectures – Adopting and evolving a product-line approach*; Addison-Wesley; London; 2000; ISBN 0-201-67494-7.

[Briand, *et al.*, 1998]
Briand, L.C., S.J. Carrière, R. Kazman and J. Wüst; *A Comprehensive Framework for Architecture Evaluation*; International Software Engineering Research Network Report ISERN-98-28; 1998.

[Brombacher, 1992]
Brombacher, A.C.; *Reliability by Design*; John Wiley & Sons; Chicester; UK; 1992.

[Brooks, 1995]
Brooks, F.P.; *The Mythical Man-month, Essays on Software Engineering, 20th Anniversary Edition*; Addison-Wesley; 1995.

[Brownsword and Clements, 1996]
Brownsword, L. and P. C. Clements; *A Case Study in Successful Product Line Development*; Technical report CMU/SEI-96-TR-016 ESC-TR-96-016; Pittsburgh, PA 15213-3890; Carnegie Mellon University; 1996

[Chappell, 1996]
Chappell, D.; *Understanding ActiveX and OLE – a Guide for Developers and Managers*; Redmond, Washington; Microsoft Press; 1996.

[Clements, 1994]
Clements, P. C.; *From Domain Models to Architectures*; Workshop on Software Architecture; USC Center for Software Engineering; LA; 1994;
Url:http:www.sei.cmu.edu/technology/architecture/projects.html

[Clements, *et al.*, 1995]
Clements, P. C., L. Bass, R. Kazman and G. Abowd; *Predicting Software Quality By Architecture-Level Evaluation*; Fifth International Conference on Software Quality; Austin, Texas; 1995;
URL:http://www.sei.cmu.edu/technology/architecture/projects.html

[Clements and Northrop, 1996]
Clements, P. C. and L. M. Northrop; *Software Architecture: an Executive Overview*; Technical Report: CMU/SEI-96-TR-003; Carnegie Mellon University; Pittsburg, PA; Feb. 1996.

[Clements and Northrop, 1999]
Clements, P. C. and L. M. Northrop; *A Framework for Software Product Line Practice – Version 2.0*; Carnegie Mellon University; Pittsburg, PA; July 1999.

[Clements and De la Puente, 1998]

Clements, P.C. and J.A. de la Puente; "Session 4: Analysis of Software Architectures"; *Development and Evolution of Software Architectures for Product Families – Second International ESPRIT ARES Workshop, Las Palmas de Gran Canaria, Spain, Feb. 1998, Proceedings*; F. van der Linden (Ed.); LCNS 1429; Springer-Verlag; Berlin 1998; ISBN 3-540-64916-6.

[Clements, 2000]

Clements, P.; *Active Reviews for Intermediate Designs*; Technical Note; CMU/SEI-2000-TN-009; Carnegie Mellon University; Pittsburg, PA; Aug. 2000

[Cockburn, 1998]

Cockburn, A.; *Surviving Object-Oriented Projects – A Managers Guide*; Reading MA; Addison Wesley; 1998; ISBN 0-201-49834-0.

[Crosby, 1979]

Crosby, B.P.; *Quality is Free – the art of making quality certain*; New York; McGraw Hill; 1979.

[Danko and Prinz, 1989]

Danko, G. and F. Prinz; *A Historical Analysis of the Traditional Product Development Process as a Basis for an Alternative Process Model*; EDRD report; Carnegie Mellon University; 1989.

[DeBaud and Schmid, 1999]

DeBaud, J-M., and K. Schmid; "A Systematic Approach to Derive the Scope of Software Product Lines"; *Proceedings International Conference on Software Engineering 1999 – Preparing for the software century*; ACM Press; New York; 1999; ISBN 1-58113-074-0.

[DeMarco, 1979]

DeMarco, T.; *Structured Analysis and System Specification*; Prentice-Hall; 1979.

[DeMarco, 1995]

DeMarco, T.; "On Systems Architecture"; *Proceedings of the 1995 Monterey Workshop on Specification Based Software Architectures*; 1995.

[DeMarco and Lister, 1987]

DeMarco, T. and T. Lister; *Peopleware*; New York, NY; Dorset House Publishing Co.; 1987.

[Demming, 1982]

Demming, W.E.; *Quality, Productivity and Competitive Positio*n; MIT Press; Cambridge MA; 1982.

[DICOM, 2000]

The Digital Imaging and Communication in Medicine (DICOM) Standard; NEMA PS 3.X; 2000; National Electrical Manufacturers Association (NEMA) Publication Sales; 1300 N. 17th Street, Suite 1847; Rosslyn, VA. 22209; USA; 2000.

[Dolan, *et al.*, 1998]

Dolan, T., R. Weterings, J.C. Wortmann; "Stakeholders in Software-system Families"; *Development and Evolution of Software Architectures for Product Families – Second International ESPRIT ARES Workshop, Las Palmas de Gran Canaria, Spain, Feb. 1998, Proceedings*; F. van der Linden (Ed.); *LNCS*; Vol. 1429; Springer-Verlag; Berlin; 1998; ISBN 3-540-64916-6.

[Dolan, *et al.*, 2000]

Dolan, T., R. Weterings, J.C. Wortmann; "Stakeholder-centric assessment of product family architecture – practical guidelines for information system interoperability and extensibility"; *Software Architectures for Product Families – International Workshop IW-SAPF-3, Las Palmas de Gran Canaria, Spain, March 2000, Proceedings*; F. van der Linden (Ed.); *LNCS*; Vol. 1951; Springer-Verlag; Berlin; 2000; ISBN 3-540-41480-0.

[Erens, 1996]

Erens, F.J.; *The Synthesis of Variety – Developing Product Families*; Ph.D. Thesis; University Press TU Eindhoven; Eindhoven; 1996; ISBN 90-386-0195-6.

[Erens and Verhulst, 1997]

Erens, F.J. and K. Verhulst; "Architectures for Product Families"; *Computers in Industry*; Vol. 33, no. 2-3, pp. 165-178; 1997.

[Ellis, *et al*., 1997]

Ellis, J., R.F. Hilliard, P.T. Poon, D. Rayford, T.F. Saunders, B. Sherlund and R.L. Wade; "Toward a Recommended Practice for Architectural Description"; *Proceedings of the second IEEE International Conference on Engineering of Complex Computer Systems*; pp 408-413; Los Alamitos CA; IEEE Computer Society Press; 1996.

[Fagan, 1986]

Fagan, M.E.; "Advances in Software Inspections"; *IEEE Transactions on Software Engineering*; Vol. 12, no. 7; pp 744-751; July 1986.

[Fowler and Scott, 1997]

Fowler, M., and K. Scott; *UML Distilled – applying the standard object modelling language*; Addison Wesley; 1997.

[Fritz, 1997]

Fritz, R.; "The Programmer ….Fishing?"; *IEEE Software*; Vol. 14, No. 5; September-October 1997; pp 8-9.

[Gacek, *et al.*, 1995]

Gacek, C., A. Abd-Allah, B. Clark, B. Boehm; "On the Definition of Software System Architecture"; *ICSE 17 Software Architecture Workshop*; April 1995.

[Gamma, *et al.*, 1995]

Gamma, E., R. Helm, R. Johnson and J. Vlissides; *Design Patterns*; Reading, MA; Addison-Wesley; 1995.

[Garlan, 1995]

Garlan, D. (Ed.); "First International Workshop on Architectures for Software Systems Workshop Summary"; *ACM Software Engineering Notes, Vol. 20, No. 3*; July 1995.

[Garlan and Perry, 1995]

Garlan, D. and D. Perry (Eds.); "Special Issues on Software Architecture"; *IEEE Transactions on Software Engineering*; 21(4); pp 269-386; 1995.

[Garlan and Shaw, 1993]

Garlan, D. and M. Shaw; "An introduction to Software Architecture"; *Advances in Software Engineering and Knowledge Engineering, vol I*; Ambriola and Torura (Eds.); Singapore; World Scientific Publishing Company; 1993.

[Gilb, 1988]

Gilb, T.; *Principles of Software Engineering Management*; Wokingham England; Addison-Wesley; 1988; ISBN 0-201-19246-2.

[Grady, 1992]
Grady, R.; *Practical Software Metrics for Project Management and Process Improvements*; Englewood Cliffs; NJ; Prentice Hall; 1992; ISBN 0-13-720384-5.

[Grady and Caswell, 1987]
Grady, R. and D. Caswell; *Software Metrics: Establishing a Company-Wide Program*; New Jersey; Prentice Hall; Englewood Cliffs; 1987.

[Van den Hamer, *et al.*, 1998]
Hamer, P. van den, F. van der Linden, H. Obbink, R. van Ommering; ARES Workshop Feb. 98 – Trip Report; Internal Philips-Research Document – CONFIDENTIAL; March 1998.

[Van den Hamer and Lepoeter, 1996]
Hamer, P. van den, and K. Lepoeter; "Managing Design Data: The 5 Dimensions of CAD Frameworks, Configuration Management, and Product Data Management"; *Proceedings of the IEEE;* vol 84, No. 1; Jan. 1996.

[Hammer, 1996]
Hammer, D. K.; "IT-Architecture: A Challenging Mix of Aspects"; *Workshop on Engineering of Computer-Based Systems (ECBS'96) – Architecture Metrics and Measurements*; New Orleans, Louisiana, USA; Oct. 1996.

['t Hart, 1997]
Hart, M. 't; *Designing IT-support for Professionals*; Ph.D. Thesis; Eindhoven University Press; Eindhoven; 1998; ISBN 90-386-0385-1.

[Hauser and Clausing, 1988]
Hauser, J.R., and D. Clausing; "The House of Quality"; *Harvard Business Review*; May/June, 1988; pp. 63-73.

[Hegge, 1995]
Hegge, H.; *Intelligent Product Family Descriptions for Business Applications*; Ph.D. Thesis; Eindhoven University Press; Eindhoven; 1995; ISBN 90-386-0491-2.

[Hofmeister, *et al.*, 2000]
Hofmeister, C., R. Nord and D. Soni; *Applied Software Architecture*; Reading, MA.; Addison-Wesley; 2000; ISBN 0-201-32571-3.

[Humphrey, 1989]
Humphrey, W.S.; *Managing the Software Process*; Reading MA; Addison-Wesley; 1989.

[IEEE 1471, 2000]
IEEE Recommended Practice for Architecture Description; IEEE Std. 1471; 2000

[ISO-9126-1]
ISO/FCD 9126-1 – Information Technology – Software Quality Characteristics and Metrics – Part 1: Quality characteristics and sub-characteristics.

[ISO-9126-2]
ISO/IEC 9126-2 – Information technology – Software Quality Characteristics and Metrics – Part 2: External metrics

[Jackson, 1995]
Jackson, M.; *Software Requirements and Specifications – a lexicon of practice, principles, and prejudices*; Wokingham England; ACM Press; 1995; ISBN 0-201-87712-0.

[Jacobson, *et al.*, 1992]
Jacobson, I., M. Christerson, P. Jonsson and G. Overgaard; *Object Oriented Software Engineering – A Use Case Driven Approach*; Reading MA; Addison-Wesley; 1992.

[Jacobson, *et al.*, 1997]

Jacobson, I., M. Gris, P. Jonsson; *Software Reuse – Architecture, Process, and Organisation for Business Success*; NY; ACM Press; 1997; ISBN 0-201-92476-5.

[Jacobsen, *et al.*, 1999]

Jacobsen, I., G. Booch, J. Rumbaugh; *The Unified Software Development Process*; Reading MA; Addison Wesley; 1999.

[Jazayeri, *et al.* 2000]

Jazayeri, M., A. Ran, and F. van der Linden; *Software Architecture for Product Families – Principles and Practice*; Boston; Addison-Wesley; 2000; ISBN –0-201-69967-2.

[Jönsson, 1991]

Jönsson, S.; "Action Research"; *Information Systems Research: Contemporary Approaches And Emergent Traditions*; H.E. Nissen, H.K. Klein, R. Hirschheim (Eds.); Elsevier Science Publishers B.V. (North-Holland); 1991.

[Kazman, *et al.*, 1996]

Kazman, R.,G. Abowd, L. Bass, P. Clements; "Scenario-Based Analysis of Software Architecture"; *IEEE Software*; Vol. 13, No. 6; Nov. 1996; pp 47-57.

[Kazman, *et al.*, 1998]

Kazman, R., M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carrière; "The Architecture Tradeoff Analysis Method"; *Proceedings of the 4th International Conference on Engineering of Complex Computer Systems*; Aug. 1998.

[Kazman, *et al.*, 1999]

Kazman, R., M. Barbacci, M. Klein, S.J. Carrière, and S.G. Woods; "Experience with Performing Architecture Tradeoff Analysis"; *Proceedings of ICSE99; New York*; ACM Press; 1999, pp. 54-63.

[Kazman, *et al.,* 2000]

Kazman, R., S.J. Carrière, and S.Gscipline of Scenario-based Architectural Engineering"; *Annals of Software Engineering*; Vol. 9; 2000.

[Kennedy, 1979]

Kennedy, M.M.; "Generalizing from Single Case Studies"; *Evaluation Quarterly*; 3 pp. 661-678;1979

[Kontio, 1996]

Kontio, J.; "A Case Study in Applying a Systematic Method for COTS Selection."; *Proceedings of the International Conference on Software Engineering*; Berlin; 1996.

[Kruchten, 1995]

Kruchten, P.; "The 4+1 View Model of Architecture"; *IEEE Software*; Vol. 12 (6);Nov. 1995; pp. 42-50.

[Kusters, *et al.*, 1999]

Kusters, R., R. van Solingen, and J. Trienekens; "Identifying Embedded Software Quality: Two Approaches"; *Quality and Reliability Engineering International*; Vol. 15 pp. 485-492; 1999.

[Lassing, *et al*., 1999]

Lassing, N., D. Rijsenbrij, and Hans van Vliet; "The goal of software architecture analysis: confidence building or risk assessment"; *Proceedings, Benelux Software Conference*; Amsterdam; 1999.

[Lassing *et al*., 2000]

Lassing, N., D. Rijsenbrij, and H. van Vliet; *Viewpoints on Modifiability*; Technical Report; Vrije Universiteit Amsterdam; 2000.

[Van der Linden, 1998]

Linden, F. van der (Ed.); *Development and Evolution of Software Architectures for Product Families – Second International ESPRIT ARES Workshop, Las Palmas de Gran Canaria, Spain, Feb. 1998, Proceedings*; *LNCS*, Vol. 1429; Springer-Verlag; Berlin; 1998; ISBN 3-540-64916-6.

[Van der Linden, 2000]

Linden, F. van der (Ed.); *Software Architectures for Product Families – International Workshop IW-SAPF-3, Las Palmas de Gran Canaria, Spain, Mar. 2000, Proceedings, LNCS*, Vol. 1951; Springer-Verlag; Berlin; 2000; ISBN 3-540-41480-0.

[Van der Linden and Müller, 1995]

Linden, F. van der, and J. K. Müller; "Creating Architectures with building Blocks"; *IEEE Software*; Nov. 1995; pp. 51-60.

[Macaulay, 1993]

Macaulay, L. A.; "Requirements as a Cooperative Activity"; *Proceedings of IEEE International Symposium on Requirements Engineering San Diego, CA*; IEEE Computer Society Press; Los Alamitos CA; 1993.

[Macaulay, 1996]

Macaulay, L. A.; *Requirements Engineering*; Springer-Verlag; London; 1996; ISBN 3-540-76006-7.

[Maier, 1996]

Maier, M. W.; "Architecting Principles for systems-of-systems"; *Proceedings of the Sixth Annual International Symposium of the International Council on Systems Engineering*; 1996

[McIlroy, 1968]

McIlroy, M.D.; "Mass produced software components"; Proceedings of NATO conference on Software Engineering; Garmish, Germany; Oct., 1968; Naur and Randall (Eds.); NATO Science Committee; Brussels; 1969 (Published as a book in 1976.).

[Meyer and Lopez, 1995]

Meyer, M. H. and L. Lopez; "Technology Strategy in a Software Products Company"; *Journal of Product Innovation Management*; Vol. 12; pp. 294-306; 1995.

[Miller, 1956]

Miller, G.; "The Magical Number Seven Plus or Minus Two"; *Psychological Review*, 63; 1956; pp81-97.

[Morris and Ferguson, 1993]

Morris, C.R. and C.H. Ferguson; "How Architecture Wins Technology Wars"; *Harvard Business Review*; March-April 1993. pp 86-96.

[Obbink, *et al.*, 1998]

Obbink, H., P.C. Clements, F. van der Linden; "Introduction"; in [Van der Linden, 1998]

[Perry and Wolf, 1992]

Perry, D.E. and A.L. Wolf; "Foundations for the Study of Software Architecture"; *ACM Software Engineering Notes*; Oct. 1992; pp. 40-52.

[Pine, 1993]
Pine II, B. Joseph.; *Mass Customisation: the new Frontier in Business Competition*; Harvard Business School Press; Boston MA; 1993; ISBN 0-87584-372-7.

[Plossl and Welch, 1979]
Plossl, G.W., and W.E. Welch; *The Role of Top Management in the Control of Inventory*; Reston Publishing company; Reston Virginia; USA; 1979.

[Porter, 1980]
Porter, M. E.; *Competitive Strategy: Techniques for Analyzing Industries and Competitors*; The Free Press; 1980.

[Postema, 1998]
Postema, H.; "Architecture Assessment: Approach and Experiences"; *Proceedings SPI98; The International Conference on Software Process Improvement*; Monte Carlo; 1998.

[Punter, 2001]
Punter, T.; *Goal Oriented Evaluation of Software*; (Dutch); Eindhoven; University Press TUE; 2001; ISNB 90 386 0863 2.

[Raymond, 1998]
Raymond, E.S., *The Cathedral and the Bazaar*; 1998;
URL: http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html .

[Rechtin, 1991]
Rechtin, E.; *Systems Architecting – Creating And Building Complex Systems*; Englewood Cliffs NJ; Prentice-Hall; 1991; ISBN 0-13-880345-5.

[Rechtin and Maier, 1997]
Rechtin, E., M.W. Maier; *The Art of Systems Architecting*; Boca Raton; CRC Press; 1997; ISBN 0-8493-7836-2.

[Rumbaugh, *et al.*, 1999]
Rumbaugh, J., I. Jacobsen, G. Booch; *The Unified Modeling Language Reference Manual*; Reading MA; Addison Wesley; 1999.

[Sanchez, 1996],
Sanchez, R.; "Strategic Product Creation: Managing New Interactions of Technology, Markets, and Organisations"; *European Management Journal*, Vol. 14 no. 2 Apr. 1996; pp121-138; Elsevier Science Ltd.; 1996.

[Schön, 1983]
Schön, D.A.; *The Reflective Practitioner*; Basic Books; New York; 1983.

[SEI product line, web]
Software Engineering Institute; *Product Line Practice 1999*;
URL: http://www.sei.cmu.edu/plp/index.html

[Shaw and Garlan, 1996]
Shaw, M. and D. Garlan; *Software Architecture – Perspectives on an Emerging Discipline*; Upper Saddle River; NJ; Prentice-Hall; 1996; ISBN 0-13-182957-2.

[Simon, 1981]
Simon, H.A.; *The Sciences of the Artificial*; Cambridge MA; The MIT Press; 1981; ISBN 0-262-19193-8.

[Soni, *et al.*, 1995]
Soni, D., R.L. Nord and C. Hofmeister; "Software Architecture in Industrial Applications"; *Proceedings of the ICSE '95, The 17th International Conference on Software Engineering*; R. Jeffrey and D. Notkin (eds.); NY; ACM Press; 1995.

[Van Strien, 1986]
Strien, P.J. van; *Practice as Science*; Assen; Van Gorcum; 1986.

[Szyperski, 1998]
Szyperski, C.; *Component Software – Beyond Object-Oriented Programming*; Harlow England; Addison-Wesley; 1998.

[Verrijn-Stuart, 1989]
Verrijn-Stuart, A.; "Some Reflections on the Namur Conference on Information System Concepts"; *Proceedings of the IFIP TC 8 / WG 8.1 Working Conference on Information System Concepts: An In-depth Analysis*; E.D. Falkenburg, P. Lindgreen (Eds.); North-Holland; Amsterdam; 1989; ISBN: 0-444 88323 1.

[Verschuren and Doorewaard, 1995]
Verschuren, P. and H. Doorewaard; *Het Ontwerpen van een Onderzoek* (Eng. Research Design); Lemma BV; Utrecht; 1995; ISBN 90-5189-546-1.

[Warmer and Kleppe, 1996]
Warmer, J. and A. Kleppe; *Praktisch OMT* (Eng. Practical OMT); Addison-Wesley; Nederland; 1996; ISBN 90-6789-776-0.

[Weiss and Lai, 1999]
Weiss, D. M., and C.T.R. Lai; *Software Product-Line Engineering – A Family-Based Software Development Process*; Addison-Wesley; 1999; ISBN 0-201-69438-7.

[Womack, *et al.*, 1991]
Womack, J.P., D.T. Jones, D. Roos and D. Sammons Carpenter; *The machine that Changed the World: the Story of Lean Production*; HarperPerennial; 1991; ISBN 0-06-097417-6.

[Wortmann, *et al.*, 1997]
Wortmann, J.C., D.R. Muntslag, and P.J.M. Timmermans (Eds.); *Customer Driven Manufacturing*; London; Chapman & Hall; 1997; ISBN 0-412-5703-0-0.

[Yin, 1994]
Yin, R.K.; *Case Study Research: Design and Methods*; Beverly Hills, CA; Sage Publications; 1994.

[Zachman, 1987]
Zachman, J.A.; "A Framework for Information Systems Architecture"; *IBM Systems Journal*; Vol. 26, No. 3, pp 276-292; 1987.

[Van der Zwaan, 1990]
Zwaan, A.H. van der; *Organisatie-onderzoek: leerboek voor de praktijk: het ontwerp van onderzoek in organisaties* (Eng. Organisational Research in Practice: the design of research in organisations); Von Gorcum; Assen; 1990.

[Van der Zwaan, 1998]
Zwaan, A.H. van der; *From case to case: discovery or validation? On the incomplete utilisation of case-studies*; (Dutch); NOBO; 1998.

[Zwegers, 1998]

Zwegers, A.; On Systems Architecting – a study in shop floor control to determine architecting concepts and principles; Ph.D. Thesis; Eindhoven University Press; Eindhoven; 1998; ISBN 90-386-0699-4.

Appendix A

Case study reports and analysis

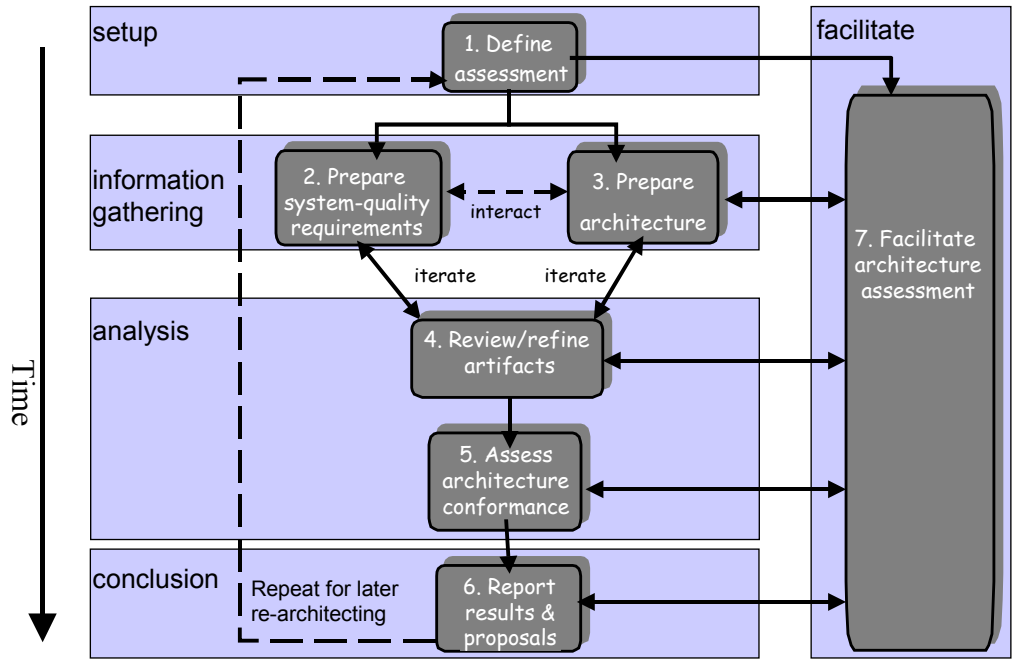


Figure A-1: Overall assessment method process

A.1 Medical case study – report

The case studies will be described based on the method process as shown in Figure A-1 above. Comprehensive explanation of the activities can be found in Chapter 5; the report here will only highlight interesting conformance or deviation from the schema. The overall context and aims of the medical case have been briefly described in section 4.3 of Chapter 6.

A.1.1 Define assessment

This step followed the overall schema as outlined in Chapter 5. An interesting aspect of this small case was that, because of the tight schedule, the architect got involved only when the stakeholders had first agreed to use the method and were clear on what they wanted to assess. So the architect was introduced to the method, with the requirements proposal (in one-line form) at the rank/select change-cases meeting.

Preliminary versions of the ***family-feature-map*** and ***migration-map*** were made during the assessment, but these were not reviewed or approved by general business management – the argument was that this was a family-management activity which was outside the scope of this pre-development, viewing-component project (code named "VIEW" here). The ***family-context-diagram***, and ***change-case-guidelines*** were used as intended and, as expected, provided motivation to the stakeholders in generating change-cases – the ***family-context-***

diagram did not exist (on paper) and was made during this step. The ***ranking-criteria*** were also used to prioritise the set of issues to be assessed. Interestingly, although a mix of extensibility and interoperability issues were raised in the initial stakeholder brainstorm using the topology diagram; it was decided for reasons of easing-the-introduction to concentrate on a set of ***extensibility*** requirements only for the initial assessment cycle. Further (without motivation from the facilitator) the assessment-sponsor naturally expected that there would be repeat sessions to address the remaining requirements. The requirements selected for assessment dealt with extending the viewing-component to accommodate mixed-monitor formats for display (see change-case in Appendix B)

A.1.2 Prepare Requirements

This step also followed the overall schema as outlined in the Chapter 5, with the notable issue being the restricted application of the family-related techniques. The ***change-case template*** was used and there are two main issues here:

- The template was seen as a positive motivation for the stakeholder; when forced to complete the extensibility-context section the stakeholder was triggered to consider legacy issues (important for family-oriented development) resulting in the modification of a requirement
- The template was modified in use; where instead of repeating the preamble and motivation for each change-case, this was kept common and the individual cases illustrating the requirements were portrayed in the extensibility-detail section (see Appendix B).

A.1.3 Prepare Architecture

The architecture was already documented with the ongoing prototype implementation. This architect didn't start preparing his architecture presentation until after the Review/refine session where the stakeholders presented their refined change-cases (see Appendix B). This close iteration with the refining step is anticipated in the method (see Figure A-1 for method overview).

A.1.4 Review/refine artefacts

The biggest departure from the method was in this step; where, instead of a common meeting, the stakeholders reviewed their requirements with the facilitator in the absence of the architect. However, the architect had received the draft requirements by email and had a chance to raise comments outside of the meeting. This deviation was justified by tight project-schedules and the perceived low risk of requirements-architecture miscommunication. The stakeholders had completed 70% of the requirements unaided, with the facilitator doing some post-review editing to improve readability prior to passing the refined requirements to the architect. Regarding the architecture representation; the *UML notation* was adopted by the project for design visualisation, and the architect selected relevant parts of the architecture to

illustrate the effects of the proposed changes (see Appendix B). The *logical view* (in [Kruchten, 1995] terms) of the software modules was used to describe main problem elements and the *development view* referred to when discussing the impact of the changes in the source code.

A.1.5 Assess Architecture conformance

The architect was leading in this step, and had prepared his presentation based on explaining how the underlying (reused by new viewing component) application platform provided the current functionality with respect to screen layout and image display. The main conclusion was that the 4 change-cases presented had similar, large effects on the architecture, and that if changes were attempted then the changes should probably be done together. Separate details of the effects per change-case (including the ***interaction/sensitivity metrics***) were not prepared in advance, and this refining of the presentation was done during the meeting, which took some time. There was some brainstorming done on possibilities to resolve the issues; this encouraged discussion, increased insight, and generated many alternatives; some of which were outside the architecture.

A.1.6 Report Results and proposals

The main assessment findings were verbally summarised at the wrap-up of the actual conformance meeting. The written report records these (see the ***requirements-architecture interaction metric*** in Appendix B, Table B-1) and the motivating factors from the architecture presentation and subsequent discussion. The ***conformance-statement template*** provided by the method was used to structure the report. Consultation with the architect was necessary to accurately reflect the technical accuracy of the findings and recommendations. During the review of the first draft of the conformance statement prepared by the facilitator, the architect re-examined his original impact analysis (from the assessment meeting) and came to a more efficient solution to addressing the change-cases. Such new architectural insights obtained by the architect when considering changes is the motivation for such assessments. An important overall conclusion from this case was the need to critically establish the business and application case for the changes because of their expected impact on the core family infrastructure.

A.1.7 Facilitate architecture assessment

The facilitation activity progressed in parallel with the main assessment activities and served to ensure the smooth implementation of these. A significant part of facilitation is to establish and maintain motivation, and momentum during the assessment; this section presents experiences with these and other implementation aspects of the medical case.

Regarding initiating the assessment; an important innovation was the fact that the assessment-sponsor emphasised the requirements analysis aspect of the activity, using the architecture to facilitate a requirements discussion, rather than as a “judgement” on the architecture that had

addressed the aims of its project brief. This was a preventive action to mitigate any eventual reluctance on the part of team members to having their work assessed/judged. This is particularly so with architects who hold senior positions, and are well regarded, in the organisation. In this case, however, the team was very receptive to the assessment but the requirements-emphasis helped to increase the architect's buy-in as he wanted to see some documented rationale relating to the particular requirements under consideration. The general point is that significant "people-management" must be brought to bear in order to ensure that no threat is manifest or perceived with the architecture assessment. This will vary with local conditions but its occurrence should be planned for.

The assessment group was very small: 3 project-people (business, application stakeholders plus architect) and 1 external facilitator. There was also some duplication of roles in individuals: the same individual played the role of ***assessment-sponsor*** and ***business-stakeholder***; the ***architect*** represented himself and the ***development-stakeholder***. This meant that assessment management was simplified, as contacts were few, meetings could be organised on short-notice. Also because the project-team was very small (the assessment participants represented 80% of the resources) group communication was easy because people had a close, stable operational history.

The total throughput time for the assessment was 3 weeks, the throughout time between requirements review and architecture conformance assessment was one week – this gives the architect adequate time to prepare his presentation in response to the requirements. The effort expended was lower than the average indicated in Chapter 5, due to the small, tightly knit project team, and focused assessment scope.

A.1.8 Overall Impressions

The participants were surveyed as to their experiences and opinions on the exercise. In general they were positive on the success of the method in giving them (the development team) the opportunity to explore, in a structured way, some important system-quality requirements. The architect was also positive about the contribution the method towards establishing and communicating architecture rationale, and for encouraging extra research into determining the business-case for the proposed requirements. Participants very much see the long-term value of the exercise in terms of a repeated activity that explores specific requirements-architecture themes in an incremental way throughout the development cycle.

A.2 Medical case study – analysis

This section recaps and analyses the most important findings from the case study report. The lessons (results of the analysis) learned with respect to the design of the method techniques and general industrial implementation guidelines are extracted and provided as feedback to enrich the method design. These findings are organised into the following categories based on the main research themes as indicated in section 1 of Chapter 6:

1. Architecture assessment;
2. Family stakeholders and qualities (interoperability and extensibility);
3. Method techniques
4. Implementation guidelines

The numbers associated with the observations and lessons are references to original case study reports, and their only role in this section is that reference.

Case overview

The most notable aspects of this case with respect to method validation are:

- This was a **small case** in terms of the numbers of people involved and the scope of requirements considered.
- This case dealt with the quality aspect **extensibility** within the context of an existing **family** and looked at extending a new common-component to deal with end-user variety as a result of allowing users to extend the capabilities of their existing workspots.
- The case was the **first experience** of stakeholder-oriented architecture assessment in the group.

A.2.1 Architecture Assessment

Observation M4

The first step of the case identified more requirements than were chosen to proceed with further in this first assessment cycle. This means that "learning effects" and the introduction of new practices are big concerns for industrial stakeholders. Also brainstorming typically generates more than you can deal with in one go.

Lesson M4

- The fact that a new method is introduced to a system means not only consequences for the system but also for the method – or at least its potential for full exploitation. People will err on the side of caution and restraint both in disturbing the system and in exploiting the new method.
- The case for a repeated, incremental application of architecture assessment is strengthened when realising that initial brainstorming typically generates more "material" than can be handled in the first assessment cycle (where we try to restrict to 10 change-cases). This prepared-brainstorm effort should not be wasted and the repetition of later method phases using these results enables the reuse of the material generated. This validates the underlying motivation of the research; for repeated, incremental architecture assessment on an ongoing basis throughout the family definition and maintenance.

Observation M9

As well as exploring how to resolve the requirements through impacting the existing architecture the architecture-conformance discussion also highlighted ways to address the requirements through non-architectural solutions such as: implementation-configurations, modifying other systems.

Lesson M9
The architecture assessment discussions prompt the participants to critically examine both the architecture and its external environment, thereby increasing insight and understanding of both. This is particularly true for the non-architects.

Observation M11
An important action derived from the assessment was the need to establish a solid business-case for the proposed changes because of their relatively high impact on the family architecture.
Lesson M11
Re-examination of the real business case underlying the proposed changes is to be expected from architecture assessments where the costs of implementing the change are high. Although this is not surprising; it does validate the role of architecture assessments in providing a mechanism and a context with which to *invite* stakeholders to be explicit and thorough with their requirements within family development.

A.2.2 Family stakeholders and qualities (interoperability and extensibility)

Observation M2
Family-feature-map was not complete or approved during the assessment.
The case was not at system-family level and was essentially concerned with a common component, which had to provide a functionality to replace the common aspects of currently separate applications. The ***family-feature-map*** is oriented towards explicitly recording those common and variable aspects of the family; in this project the focus was on common functionality only, and that knowledge was documented in functional specifications. In theory the overall product business management should have (made) this map and allocated this project as providing a common component.
Lesson M2
It may be initially concluded that ***family-feature-map*** is something that does not come naturally to those staring-out in technology-driven (bottom-up) family development, and that it is most relevant when the development group have to actively consider both variants and commonalities.

Observation M3
The ***migration-map*** was not complete during the assessment. Again, the migration-map is typically something of most relevance for the overall family business, dealing with the product planning of the family over time, both past and future. This was perceived as outside the scope of the current project – although some of the brainstorm change-cases referred to migration issues. These changes were not pursued in this session and the need for the ***migration-map*** did not manifest.

Lesson M3

The ***migration-map*** addresses family evolution, and does not come naturally to those who are starting out (from a technology bias) in family-development exploring the possibility to build common components. It is most appropriate when there is active need to consider commercial family evolution, in a market-driven release project.

A.2.3 Method techniques

Observation M5

The change-case template motivated the stakeholder to consider extra requirements dimensions and enriched his specification

Lesson M5

The change-case template is a positive contribution to stimulating requirements generation – this conforms to the design intention.

Observation M6

The stakeholder customised the use of the change-case-template

Lesson M6

The method is a structure to help and facilitate stakeholders – low-level customisation is to be expected and should be welcomed (as stated in Chapters 2 and 5). The techniques facilitate deliverables, and should be perceived as a framework, where the result is more important than the form.

Observation M7

The group meeting with all participants to review and refine the requirements/architecture was not attended by the architect. This arose because of time pressure and the solid understanding of the requirements by the architect.

Lesson M7

Not all feedback should necessarily occur in meetings – certain clarifications needed in requirements/architecture prior to the conformance meeting can be most efficiently done via direct communication or email. The facilitator should be kept aware of progress and should be pro-active in enquiring on same.

Observation M10

While the facilitator is responsible for reporting the assessment findings; architect-feedback was needed to provide the correct technical details on the report.

Lesson M10

Architect involvement is necessary to properly document the rationale, especially in providing a textual response to the requirements and providing an architectural context to the discussion. This involvement is preferably in reviewing the technical accuracy of the findings and any recommendations, as drafted by the facilitator.

A.2.4 Implementation guidelines

Observation M1

Sponsor felt that because the project was small and almost complete introducing extra "noise" had to be well managed. First the stakeholders as a group had to support and prepare (their initial deliverable for) the method before selling it to the architect and rest of the project-team. Part of the selling-tactic was to include concerns of personal interest to the stakeholder/architect as part of the requirements. This made the assessment seem more established with the stakeholders and a part of the project

Lesson M1

If:

- the assessment represents a large proportion of unplanned work in a project;
- reluctance to assessment is expected (due to project timeframe or goals) from the architect/stakeholders;
- the group is small and likely to polarise;

then getting piece-meal acceptance from sponsor and certain stakeholders before trying to convince the whole assessment team together may be more effective.

Observation M8

The architectural effects per change-case were not prepared in advance of the architecture presentation; this cost some meeting time in organising it.

Lesson M8.

Facilitator and architect should communicate *details* on presentation plan before the architecture conformance session.

Observation M12

In order to avoid the sometimes "negative/threatening" connotations associated with the term "assessment" – the method and the exercise was pitched (including changing the presentation slides) as a requirements analysis exercise; therefore shifting the message towards requirements and the stakeholders instead of the architect.

Lesson M12

Depending on local situations – it may be appropriate to modify the "user interface" of the method so that it presents a less-judgmental face. In that case the methods true aim – of constructively facilitating a requirements-architecture dialogue to establish risk and encourage bi-partisan negotiated requirements and architecture directions may be more effectively achieved. An open term like "architecture-centric analysis method" may be more appropriate.

A.3 Planning Case study – report

This case study is implemented and described based on the method template. The graphical schemas summarising the step details, used in the medical case report, will not be repeated here. Readers are referred to Figure A-1 for pictorial summaries and Chapter 5 for comprehensive explanation of the activities. As with the medical case, the report here will

only highlight interesting conformance/deviation from the proposed method. The overall context and aims of the planning case have been briefly described in section 4.3 of Chapter 6.

A.3.1 Define assessment

This case study was organised and implemented very quickly (see section A.3.7 below), with the result that there was some deviation from the activity-meeting relationship as designed. Regarding assessment definition; the main feature was that the method introduction:

- was done by the architect (who had minimal introduction to the method);
- in a 1-to-1 setting with each stakeholder;
- combined with presenting the architecture to the stakeholders (to bring them up to speed on the project goals).

This was due to project pressure; there was not enough time to organise a dedicated group-session for method introduction. Stakeholders, therefore, did not have the benefit of group interaction with respect to shared method understanding or requirements generation. This provided a chance to test a counter-argument – there is no need to spend significant time explaining the method to the group. The later problems encountered with stakeholder understanding and acceptance of e.g. the ranking and template techniques (see section A.3.2) illustrates the falsification of this counter-argument [Yin, 1994].

As with the medical case, this relatively young, technology-driven, pre-development project had not yet done extensive business-planning, so the ***family-feature-map***, ***migration-map*** were not available for the assessment team prior to the assessment, and only very-draft forms used during the assessment. The sponsor actually intended the assessment to be used as a vehicle to gather stakeholder input and buy-in to the project, so it is envisaged that the business-aspects of the family will receive more attention hereafter. The ***system-topology-diagram***, and ***change case guidelines*** were provided as part of the architecture documentation, and were used by stakeholders.

The sponsor (also the business manager stakeholder) agreed to set ***interoperability*** as the theme of the assessment as this was a central issue in the planning project; which sought to enable the family to be re-engineered to uncoupled, application-components co-operating across business locations.

A novel aspect of this assessment definition was the inclusion of an *actual customer-representative* (from an important client) among the participants. This added credibility to the assessment, and also provided valuable input to the requirements.

A.3.2 Prepare requirements

Generation and preparation of requirements were done individually and in-parallel by stakeholders, rather than sequentially as prescribed. Each stakeholder generated his/her requirements (4 or 5 one-line change-cases) after the introduction of the method and

architecture, with the advice to think about interoperability. The family context was present in the sense that the stakeholders had an intuitive knowledge of the existing system-family and also the brief of the project was clearly concerned with providing a family-infrastructure (see also case description in Chapter 6).

When the stakeholders did apply the change-case-template, they were almost universal in their lack of appreciation for it. This contrasts with the previous case, and can be partially explained by the fact that stakeholders did not have a thorough introduction to the template.

The "requirements meeting" was the first collective meeting of the stakeholders with the architect and external-facilitator. The meeting was scheduled for two hours to hear and select the change-cases. In fact this activity should have happened in the definition step, but did not. Stakeholders (8 of them) were asked to briefly (5 minutes) present 3 of their change-cases to the group for initial reaction so that enough information could be shared to prioritise, reduce the number to the maximum of 7 advocated by the method.

First-time effects were obvious, as some (25%) of the stakeholders did not have one-line change-cases prepared – but a more general description of their particular concern. Again this can probably attributed to the "light" introduction provided.

The lack-of-preparedness, coupled with the size of the group (8 stakeholders, architect, assessment sponsor), and the fact that it was the first meeting, may also have contributed to the fact that discussion was extensive and time-management during the meeting was difficult. Overall presentation and discussion (all necessary activities) consumed 75% of the time – leaving a short time for change-case grouping, prioritisation and selection. During refinement this meant that stakeholders included almost all aspects of their group-of-change-cases in the single change-case allocated to them. Recommendations on this issue are provided in the analysis section below.

In the end, of the 24 change-cases proposed, 8 re-formulated change-cases were chosen to be carried-through to the architecture presentation (see Appendix C below). In order to maintain stakeholder buy-in and ensure a broad coverage, all stakeholders were associated with at least one change-case; sometimes as sole author (5 stakeholders) and sometimes in a co-authoring role (6 stakeholders).

A.3.3 Prepare architecture

The architect had an existing architecture document, and this had been *technically* reviewed by colleagues from within the development department. As in the previous case (and the method), the architecture customised his architecture presentation after the change-case selection and refinement.

A.3.4 Review/refine artefacts

After the requirements meeting there was no group-review planned (again, for timing reasons) prior to the architecture-conformance presentation. Stakeholders were requested to provide their refined change-cases to the architect 4 days before the presentation. Any issues were resolved bilaterally between architect and stakeholder.

A.3.5 Assess architecture conformance

The architecture-conformance meeting occurred one week after the "requirements meeting". It was a three-hour meeting, led by the architect, and organised around providing feedback per change-case to the stakeholders. The order and timing of the change-case treatments were based on architectural relevance.

Two important aspects of this meeting were:

1. The participants had changed with respect to the previous "requirements meeting": three of the original stakeholders were unavailable – but two had been replaced by colleagues; and two of those present had missed the "requirements meeting", and were then represented by others. While the stakes remain the same; and such absences are common in industry – changing the people does introduce disturbances, as illustrated shortly, and should be avoided if possible.
2. Not all stakeholders had provided written refinements of their change-cases.

The latter point is notable because:

- the missing change-cases had been allocated to the missing participants;
- a public review of the material by the group (which was not done, see A.3.4 above) prior to this architecture-conformance meeting may have been sufficient to ensure that the stakeholders provided their input, or that other authors could have been arranged. In this sense the review/refine step can act as a readiness check for the assessment meeting.
- While the architect in his presentation did address the missing requirements (based on the one-line requirements from the previous meeting); other stakeholders present felt that the (missing) requirements were not properly understood, or addressed. This highlighted the obvious gap (as far as stakeholders were concerned) between the quality of the architects response to refined versus unrefined requirements. It is postulated here that the more refined the requirements, the higher the quality of the architecture response. This would validate a basic tenet of the research; that detailed, structured, requirements prepared by stakeholders provide the best input to (and output from) architecture assessments.

The architecture description used was the architect's own "box 'n line" [Bass, *et al*., 1998] style, which approximated to the *logical view* of Kruchten [1995]. This is appropriate as in the early stages of design (as this case was) the logical, more domain-centred, concerns are dominant. This view was sufficient to describe the majority of interoperability aspects of interest to the stakeholders. Occasionally the discussion infringed on some dynamic aspects of

the co-operation, and the embellishment of the *logical view* by e.g. UML-type sequence-diagrams was useful.

The meeting itself provided the intended discussions between stakeholders and architect. An interesting fact was that all the change-cases were *indirect* (i.e. required modification of the architecture, see Appendix C below). This is best explained by the fact that the stakeholders were presented with the current state of the architecture *prior* to generating their requirements (see assessment definition description in section A.3.1 above); so they had a natural disposition to explore requirements outside of what they knew was already addressed.

The assessment also revealed that some components had very high interaction values (i.e. they were involved in almost all change-cases, see Appendix C below). This is explained by the fact that the architecture was relatively early in its life cycle; with the logical view being the dominant view used, and many "modifications" referred to enhancing the level of detail of the architecture description to make services explicit. Such feedback leading to a refinement of the architecture description is one of the published [Kazman, *et al.*, 1996] benefits of architecture assessments.

A.3.6 Report Results and proposals

The assessment findings were prepared with significant input from the architect shortly after the architecture-conformance meeting. A selection of the result summaries, e.g. ***requirements-architecture interaction metric***, is provided in Appendix C below. A critical output from the case was the identification of stakeholder-driven business requirements that need to be demonstrated in the first version of a prototype used to exercise the new architecture. Further there was an agreement to share knowledge and experience with the invited customer on some specific areas; an opportunity made possible by the assessment.

A.3.7 Facilitate architecture assessment

This case tested a light-facilitation approach; mainly due to the compressed time frame available and the resulting capacity planning challenges. The entire exercise from commitment-to-start to the architecture-conformance meeting was 2 weeks. The most important feature from a method-viewpoint were the facts that:

- the method introduction in the organisation was done by someone inexperienced in the method itself; which led to some ongoing questioning about the process throughout the exercise.
- some activities designed as occurring separately were grouped, and compressed, into a single meeting (e.g. generate change-cases + present change-cases + select). Similarly the method introduction was done 1-to-1 (rather than in a group-context) combined with the architecture presentation.

The result was that facilitation was sub-optimal and may have a distorting effect on the case results. The analysis will account for this possibility.

This was a large assessment in comparison to the medical case, with stakeholders representing:

- Sales;
- Customer;
- Site Implementation;
- Product support;
- Application integration;
- Business management;
- Development management
- Development implementation

Covering adequately the main stakeholder concerns typical of information system families as outlined in Chapter 4. Each stakeholder provided 3 change-cases and overall the assessment brought 8 change-cases through to the architecture-conformance meeting (see section A.3.5 above)

An interesting feature of this case was initial reluctance by the architect to adopt the method, because of seeming overlap with normal project-reviews. However, when the method was advocated, not as a single "architecture exam", but a means to institutionalise regular stakeholder-oriented architecture-reviews and thus improve overall architecture practice, the situation changed. The assessment had the support of the architect, who then became a process-owner and was active in organising (see the individual method-step reports previously) the assessment exercise. This is again evidence of the fact that industrial practitioners want structural, repeated measures to improve processes, as postulated early in the research (see Chapters 2 and 4).

Facilitation was much more difficult and under resourced in this case because of the numbers of stakeholders, the very short time frame, and the resultant lack of meetings. Taking such short cuts with method introduction is not advised.

The total throughput time was 3 working-weeks; similarly to the medical case the throughout time between refining the requirements and architecture conformance assessment was 4 days – this gives the architect just sufficient time to prepare his presentation in response to the requirements.

A.3.8 Overall Impressions

The reaction of the participants was similar to that of the medical case. Stakeholders appreciated being involved; and expected follow-up and repeated assessments. The business sponsor and the architect both expressed satisfaction at getting important stakeholder requirements into the project. They also wanted to see such assessments become part of the standard development process in the company. While the short timeframe meant that some

issues were not as clear as possible, some stakeholders appreciated the short time-span and quick turnaround involved in the assessment. The architect would have liked more time to prepare his presentation. All said they would repeat the exercise.

A.4 Planning case study – analysis

As in the previous case, this section recaps and analyses the most important findings from the case study report.

Case overview

The most notable aspects of this case with respect to method validation are:

- This was a **large-case** in terms of the numbers of people involved and the scope of requirements considered.
- This case dealt with the quality aspect **interoperability** within the context of an existing **family** and looked at introducing a new common-communication-infrastructure (code named "COMM" here) to deal with uncoupled-component communication, facilitating customer configurability and also allowing the development of more modular application-components.
- The case was also the **first experience** of stakeholder-oriented architecture assessment in the group.

A.4.1 Architecture assessment

Observation P9

People were in general enthusiastic that their opinion was sought – and all expected follow-up and that the process should be repeated for other architectures and indeed become part of standard development practice. Further the assessment sponsor said that he had heard new ideas and issues brought up

Lesson P9

Such assessments are great opportunities for the family business manager to establish stakeholder buy-in to a project; and they generate requirements as to what must be addressed.

Observation P11

When main parts of architecture are presented to the stakeholders as part of introduction – stakeholders are informed as to what the architecture can do – therefore it is natural that most of the change-cases will tend to stretch the architecture to what it cannot do (so most change-cases will be indirect).

Lesson P11

This is natural in the circumstances and should be expected and explained to stakeholders and architect (so that the high level of indirect change-cases does not give the impression that the architecture is weak, or make the architect dispirited that all the requirements are stretching the architecture)

Observation P13

In product development setting (as distinct from acquisition) – it may be that more requirements/change-cases are generated than can be included in one session. These are important requirements and are the main reason why the iterative, repetitive approach to assessment is advocated in this research.

Lesson P13

Repeat requirements-review and architecture-conformance sessions should be held for these requirements (enables reuse of method and architecture presentations); plus provides an analogy to incremental development for the management of system qualities.

Observation P14

During architecture-conformance session the stakeholders hear plans about the changes that should/could be made. It is important to them (they express this) that they receive progress reports as to how the progress is actually proceeding –

Lesson P14

Stakeholders expect repeated architecture conformance-sessions (the requirements are now known/ "fixed" and all preceding steps can be reused) – where the architect shows how he has converted the previously indirect change-cases into direct change-cases. This acts to provide architecture feedback to stakeholders and provides a mechanism to iteratively check that architectural solutions are aligned to stakeholder expectations.

Observation P18

A couple of stakeholders and architect commented on the fact that *prepared* input was good (the main point of this approach is that stakeholders get time and techniques to prepare their own input).

Lesson P18

The core method-idea of stakeholder taking more time, responsibility for structuring their requirements seems to be recognised as useful by the participants.

Observation P19

Stakeholders were concerned in establishing if you were complete enough and had established the correct priority.

Lesson P19

The fact is that assessment is like testing – you can never be formally complete – you just hope to add value and do what you can until the value added decreases [Kazman, *et al.*, 1996]. This approach must be made clear up front as part of the method scope/limitations.

Observation P23

Project sponsor felt that such assessments should become part of normal development process – and also the company should acquire the capability to do these assessments itself (although

with the help of *an* (1) external facilitator just to make sure that one person can be politically *naïve* during the proceedings.

Lesson P23

This supports another main idea of the research that companies want to acquire this competence/behaviour themselves so that it becomes institutionalised in their way-of-working.

A.4.2 Family stakeholders and qualities (interoperability and extensibility)

Observation P1

We did not evaluate all the techniques in the method – in particular the family-aspect of COMM was not *explicitly* clear to all stakeholders or used by them. Stakeholders did have an intuitive idea of the family, and their concentration on dealing with the variety the infrastructure must deal with was evidence of this.

Lesson P1

Family issues are difficult to incorporate explicitly and the advice here is to do a normal (non-family) assessment first to get the organisation used to such assessments, before scaling up to consider the family context.

A.4.3 Method techniques

Observation P5

The stakeholders felt that the grouping was confusing and did not achieve any purpose as they did not see the benefit – they also said that it just made the change-cases too broad.

Lesson P5

There is an open question on whether the grouping was really inefficient/incorrect or was it just badly managed (not enough clarity, time, and involvement) at the time leading to the stakeholder perception that it was useless. It should be repeated under less-constrained circumstances for fair assessment.

Observation P17

The term change-case was confusing for the stakeholders – they would have preferred to see something simple like “requirement” or “scenario”.

Lesson P17

This may have to do with the fact that in the planning case the method was not explained by the external facilitator but was combined with the architect’s one-on-one presentation. The method issues were then communicated with reduced priority, second-hand, and without the benefit of group interaction. The lessons to be learned here are to:

- Drop the term change-case and just call it requirement-case/scenario we then lose the semantics attached to the term that the illustration deals with a change to the status quo (which psychologically means that it may be indirect and doesn’t look like it’s been forgotten by the architect);

- Repeat the process with a control-group where the term (it comes from literature [Bennett, 1997]) is explained by the facilitator first hand to the stakeholder group (as advised in the method) and then see if the same feedback confusion/opposition to the term occurs.

The second approach is favoured in this research.

Observation P20

Most stakeholders (when prompted explicitly for an opinion) felt that the ***change-case template*** was not really beneficial – they felt that there must be something to structure the description but that they didn't know what to put in what sections. This contrasts with the medical-case assessment where the template structure prompted the stakeholder to think of the legacy situation and modify his change-cases.

Lesson P20

Again this can be due to the fact that:

- the template is rubbish/incomplete;
- Or again, the fact that the template was not explained to the stakeholders – or that the explanatory text was insufficient.

Considering the contradictory reaction from different cases – it looks like findings are inconclusive and again the issue needs to be re-tested in similar situations.

Observation P24

Some (2/3) stakeholders wanted to know what was a standard architectural notation in order to make the communication easier.

Lesson P24

People want a standard notation in order to make the process transferable; in this case the architect had his own (common) box 'n line style also the views used were logical (not process/physical – but verify this with architect). This author's opinion is that if you have a company standard use-it – if not then perhaps try to adopt the 4+1 of Kruchten/RUP. Forcing the method to adopt a notation is premature seeing the lack of standardisation in the industry, and the fact that this method will always be part of a larger, more critical development process with its own notations and techniques. However, this method can (and does) advise; this is sufficient (see medical case).

A.4.4 Implementation guidelines

Observation P2

The purpose of the method and its approach was not clear to *all* the stakeholders – there were questions during the session on what the role of stakeholders were, and why the assessment was being done this way.

Lesson P2

There needs to be adequate time spent to thoroughly brief the stakeholders on the method – we cannot take the type of short-cuts used this time to compress architecture/method description into a short-session which concentrates on architecture.

Observation P3

Including a customer may seem dangerous – in that they see how good/bad you are and also it may tend to orient the review to be biased to the single customer's individual situation

Lesson P3

It is good because customer presence raises the importance/credibility of the review for the organisation; and also puts demands on those to follow-up. The self-confidence needed to deal with customer expectations is also necessary to make families (where multiple customers must be managed).

Observation P4

Grouping and selecting the change-cases did not go well – there was not enough time and stakeholder involvement in doing it. It resulted in aggregate change-cases which included all members of the group – this resulted in confusion for stakeholders (why? Did we group) and extra work for the architect (do I still have to address all the individual change-cases – its just that now they are collected into one big-one?)

Lesson P4

The idea should have been to:

- collect similar (from an architecture-perspective) change-cases (look for 4/5 themes);
- select/prune from the *collected* change-cases to 1/2 that illustrate the group/aspect – only this 1 should proceed to next section – in total bring 7 forward.

This typically takes and needs more time than presenting the c-cs – but in the planning case it got much less time.

- Advice – People should record the assessment metrics e.g. # people/c-c, time spent per c-c, time spent grouping; to bench-mark and improve the planning/meeting management.

Observation P6

Some stakeholders will not be prepared and will come to discuss their problem without having a specific change-case in mind.

Lesson P6

Rather than getting bogged down in discussion advice is for facilitator to move discussion on and return to this stakeholder when he think of examples based on what the others say as an illustration.

Observation P7

Sometimes when discussing change-cases – people will get into discussions on providing the solution – while some discussion is good – care should be taken so as not to spend "problem" time on "solution" issues

Lesson P7

Facilitator should move the discussion by reminding those that the idea is to get the requirements clear and understood – not resolved!

Observation P8

It's natural that stakeholders may try to include non-assessment issues (e.g. UI, performance, and some missing functionality) in the change-cases. The idea can never be to rule out something important, the purpose of the method is to facilitate important discussions not prevent them. However such practices can dilute focus and may give the impression that anything goes

Lesson P8

Two options are open:

- say it's not really focus of assessment and probably belongs to a different/dedicated forum, so exclude it;
- if its really important – put it to a stakeholder-vote and say than one of the currently-included (for the architecture-assessment session) change-cases will have to be dropped. If no time for vote – ask the business-sponsor to make the call on what's in/out.

Observation P10

There was a mixed-meeting attendance (i.e. some people from last time were missing and their c-cs were also not refined – these change-cases posed the most difficulty during the meeting (mixed understanding, plus claims from sidelines that architect's interpretation was different than intended).

Lesson P10

It is important that there is continuity in the meeting composition – this ensures commitment to refine the c-c's on part of the stakeholder and also avoids others assuming the missing stakeholders role during the architecture-assessment session. However, where there was a clear replacement planned – this went well because these stakeholders co-ordinated off-line.

Observation P12

It may be that in trying to clarify discussion that the facilitator (especially if within the company/group or industry) is tempted to express opinions on the requirements or architecture.

Lesson P12

Stakeholders may be very sensitive to this and the role of facilitator should be confined to method and meeting management. Facilitator should be conscious of not intruding on stakeholder/architect roles at all times and even prime a shadow-facilitator to quietly kick-him/her if necessary!

Observation P15

Having large numbers of stakeholders has benefits (broad input/buy-in) – but having large numbers of stakeholders (8) means that meeting management becomes difficult – especially as stakeholders do not like, or have time for, 3/4-hour meetings. There is also the issue that

having *n* stakeholders may also mean that one has to have *n* change-cases – otherwise the stakeholder feels that he has (publicly) lost out.

Lesson P15

This is a challenge to meeting management – two strategies are:

- have a large number of stakes but combine multiple stakes in a single individual – at least that way the debate time per person may be maintained but that per stake will be reduced. It will also help in change-case pruning because the chances that a person loses all his change-cases in the review-session is smaller (even though a stake may lose change-cases);
- if the number of people is unavoidably large (no obvious way to combine stakes) one way to combat some stakeholders losing their change-cases is to look for repetitions/similarities in the review phase and group the stakeholders accordingly so that the "jilted" stakeholders are partly responsible for the remaining "compromise" change-case. This may involve some redrafting of the change-cases so that a compromise text can be found. The facilitator and the architect should actively plan for this before the review meeting itself to guarantee success.

Observation P16

When many high-profile stakeholders are present it is important that meeting time is kept to a minimum (also these people typically like to have a very focused/efficient meeting). Depending on the project, company-culture gathering these people may only be possible a handful of times a year.

Lesson P16

Assessment sessions should be made as attractive and professional as possible to ensure that they will return. There should be quick follow up after meetings (minutes, follow-up planning) and not too much downtime between sessions.

Observation P21

All felt that the principle (stakeholder-centric assessment) was correct and they liked the quick-turnaround and short cycle-time (see high-profile observation earlier)

Lesson P21

Clear method/role definition and efficient focused planning are important in stakeholder acceptance – this means that the facilitation must be professional and experienced.

Observation P22

Broad input was good input for architect- but he felt that meeting and cycle-time was very short.

Lesson P22

trade-off with stakeholder wishes – again this can be reduced by better prioritisation of grouped change-cases (not well done in this assessment); by reducing number of change-cases dealt with at architecture conformance session.

Appendix B

Medical case study – sample artefacts

Some artefacts have been excluded or modified to protect confidential information.

Change Case FAAM: Multi-Monitor – version 2

Goal: End-Users can work appropriately with colour and B/W applications
Actor: End-user

Motivation

Trigger:
Specifications to-date do not foresee the use of mixed-characteristic monitors

Rationale:
Use-cases exist which seem to require the ability of the VIEW system to control multiple monitors of mixed characteristics. If the VIEW system would not be able to do so, some marketing and application action would be required to prevent wrong expectations with the end-user.

Some of the anticipated use-cases:

	Change-case	Likelyhood /Criticality (Ranking)
A	Apply Applications where colour seems to be critical in its current implementation, together with viewing images for which B/W may be more suitable	High (1)
B	View Colour Cases (eg US-colour, Doppler) or Colour-results of Applications and B/W Cases, sometimes simultaneously, sometimes sequentially	Medium/ High (2)
C	When B/W display is preferred, some users prefer some cases (CR) to be read on 2K portrait monitors instead of 1K landscape monitors	Medium/ High (3)
D	Display information from other information systems (often RIS) or other applications on VIEW e.g. via X-Windows, or installed applications like MS-Office or E-mail.	Low (4)

Specification
Brief Description:

There is a perceived business need to facilitate the fact that customers want to extend existing workspot configurations with extra and/or different (non-homogenous) monitors. In such situations the viewing-component system should:

- seamlessly accommodate the additions;
- ensure that images are shown on the "appropriate" monitor.

To accommodate the above, it is preferable that:

1. Colour monitors and B/W monitors can be combined, where B/W monitors are either 1K landscape or 2K portrait
2. That the VIEW-system puts the 'right' case on the 'right' monitor, and puts the 'right' application on the 'right' monitor

System-extensibility (upgradeability) context

It is preferable if:

- VIEW users can extend their existing system with a monitor(s) of choice at all times. This means that in the upgrade situation Spec 1. is to be changed to:

1'. Colour monitors and B/W monitors can be combined, where B/W monitors are either 1K landscape or 2K portrait, or a combination thereof.

- in the far future Image Quality can be controlled and maintained per monitor

Pre-conditions/Limitations:

- Requirements are limited by the capabilities of the NT and Unix Operating Systems and computing-platform hardware.

System-extensibility detail:

Change case A
It should be possible to combine on a workspot – the optimal viewing of images (as provided by high-resolution B/W monitors) while enabling users to employ VIEW applications which currently rely on colour as an integral part of the application (e.g. MPR)

Change case B
In a mixed colour-B/W monitor configuration (atlantis screen config. 4):

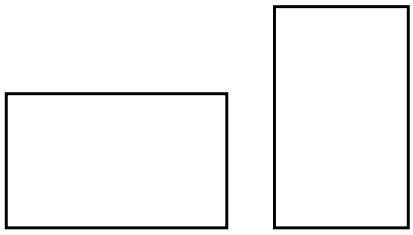
- colour cases (e.g. US);
- or mixed cases where some intermediate results (some images from the set) use colour

should display the colour-images on the appropriate colour monitor, and any b/w images on the b/w monitor.

This may have a close relationship with the DDP

Change case C
Atlantis envisaged a homogenous viewing environment with respect to resolution (1k/2k) and aspect (portrait/landscape) – however there is a marketing case where an existing 1k landscape configuration can be extended with a 2k portrait monitor in order to optimally view CR images at that workspot.

Mixed resolution/aspect monitor configuration



1K, Landscape 2K, Portrait

In this situation viewing-component should support the fact that CR cases should be viewed on the portrait monitor, while other cases should not.

This may have implications for the *continuous viewing environment* approach – and in mixed aspect situations – it makes no applicational-sense to drag landscape-oriented image-hangings to a portrait-oriented monitor and vice-versa.

Change case D
In a mixed colour-B/W configuration: if other applications (e.g. office; RIS) applications are called-up – then viewing-component should react by allowing these to superimpose themselves on the colour monitor. When viewing-component is re-awakened (e.g. by the viewing-component B/W screen receiving the focus) – then viewing-component should re-impose itself on the colour monitor.

End-note NA

Architecture views used

The UML-notation of the logical architecture view is used

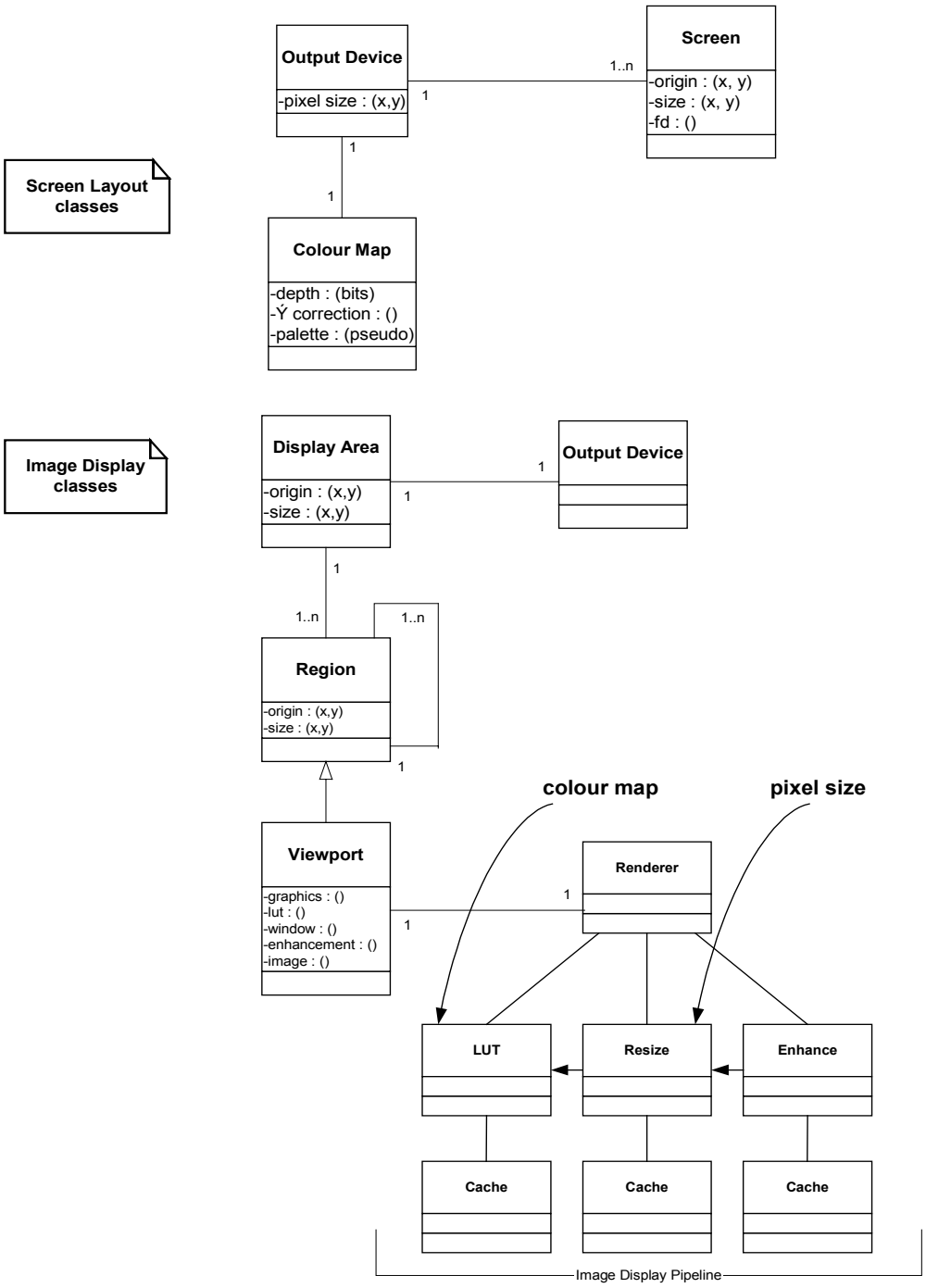


Table B-1: Results of change-case evaluation for viewing-component (interaction metric)

Ref.	issue	Direct/ Indirect	impact	No. cpts.
A	Combined colour + B&W applications	Indirect	Multiple **colour-maps** per **output-device**; change **renderer**	3
B	Combined colour + B&W: Appropriate image placement	Indirect	Multiple **colour-maps** per **output-device**; change **renderer**	3
C	Mixed resolution/aspect monitors	Indirect	Multiple **display-areas** per **output-device**; change **renderer**	3
D	Third-party applications on colour monitor	Indirect	Not addressed	?

The **renderer** and **output device** classes are affected by all change-cases (3); the **colour-map** (2) and **display area** (1) are somewhat less affected. The affects of all change cases are similar, and they should be treated together.

Appendix C

Planning case study – sample artefacts

Some artefacts have been excluded or modified to protect confidential information.

Table.C-1: Overview of change-cases selected for architecture-conformance presentation

No.	Change-case	Ranking
1	Ensure component independence with respect to installation, fault-diagnosis and repair	2
2	Components should interact with robust, asynchronous and de-coupled messaging	1
3	The system should facilitate changes to key enterprise entities and ensure that these and other operational changes are transparent/visible and propagated throughout the business-process-chain	3
4	Information about constraints, dependencies and features, about and between components, is available and used at definition time (Process Modeling), Runtime and installation time (Do not break a working configuration)	7
5	Key Component exchangeability with third-party systems must be possible	4
6	Interoperability with own and third-party components – within a multi-national context	5
7	Hub security and traceability must be explicitly supported	8
8	Development must be able to treat components as independently and de-coupled as possible	6

TableC-2: Results of change-case evaluation for COMM (interaction metric)

Ref.	issue	Direct/ Indirect	impact	No. cpts.
2	Components should interact with robust, asynchronous and de-coupled messaging	Indirect	***EME*** component needs extra services; extensive modification to existing applications (outside architecture)	1*
1	Ensure component independence with respect to installation, fault-diagnosis and repair	Indirect	***All*** components need tracing and tapping services	5

Ref.	issue	Direct/ Indirect	impact	No. cpts.
3	The system should facilitate changes to key enterprise entities and ensure that these and other operational changes are transparent/visible and propagated throughout the business-process-chain	Indirect	***All*** components need to support business-process versioning	5
5	Key Component exchangeability with third-party systems must be possible	Indirect	Third-party Modelling environments are supported directly; Third-party hubs demand extra services from our ***HUB***.	1
6	Interoperability with own and third-party application components – within a multi-national context	Indirect	***Application*** components , ***HUB*** and ***EME*** affected Design rules need clarification	4**
8	Development must be able to treat components as independently and de-coupled as possible	Indirect	***HUB***, ***sending/receiving*** components affected. Design rules needed	3**
4	Information about constraints, dependencies and features, about and between components, is available and used at definition time (Process Modeling), Runtime and installation time (Do not break a working configuration)	Indirect	EME component needs a configuration manager	1
7	Hub security and traceability must be explicitly supported	Indirect	Not addressed in meeting ***HUB***, ***sending/receiving*** components affected	3

Legend:
* - significant effort needed to migrate existing application components to the COMM architecture interface
** - needs explicit design rules to be added

The following table is the inverse of the previous table – summarising per component how many change-cases it is affected by. This highlights sensitive components.

TableC-3: architectural-component sensitivity metric

architectural component	Change-case coverage	Sensitivity Ranking
EME	5/8	2
HUB	6/8	1
Sending cpt.	5/8	2
Queuing & validation	2/8	3
Receiving cpt.	5/8	2

Index

Curriculum Vitae

Tom Dolan was born on March 31 1968 in Athlone, Ireland. After completing secondary school in the Marist College Athlone in 1986, he attended University College Galway and graduated with a B.E. degree (1st class hons.) in Industrial Engineering and Information Systems in 1990. He then joined the Computer Integrated Manufacturing Research Unit (CIMRU) in UCG as a CIM projects engineer, obtaining a Masters degree in Industrial Engineering Design and Analysis in 1992.

In November 1993, while still with CIMRU, he spent a year working in the Netherlands at TUE. Hereafter he joined the TUE as a Ph.D. student in the faculty of Technology Management. The Ph.D. research was supervised by Prof. Hans Wortmann and Prof. Dieter Hammer, and has resulted in this dissertation and some international publications.

In parallel with his research work, Tom has worked since November 1994 with Philips Medical Systems in Best. He was initially employed as a requirements engineer and helped introduce use-case-based methods in the development process. His main duties have been as a systems analyst on various radiology image and information systems products and projects, where he has worked with many international colleagues. In July 1999 Tom became responsible for interoperability architecting for the radiology information-system product family, and since November 2000 has held the position of family architect for the cardiology information-systems business.

Propositions

accompanying the Ph.D. thesis

Architecture Assessment of Information-System Families

a practical perspective

by

Thomas J. Dolan

Eindhoven, 1 June, 2001

I

Product-family architectures are shaped by business decisions, while single-product architectures are shaped by technical-implementation decisions. [Chapter 3 of this dissertation]

II

Single product development gives prominence to functional requirements over non-functional requirements; whereas this is reversed in product-family development. [Chapter 3 of this dissertation]

III

Although information systems and their development are becoming more strategically important for producers and customers, the associated strategic aspects and stakeholders are largely ignored by commercial development methods. [Chapter 2 of this dissertation]

IV

Family-based development of information systems means that more attention will have to be devoted to both *extra*- and *intra*-family interoperability. [Chapter 4 of this dissertation]

V

The recent movement in software development methods from formal design modelling towards requirements-centric practices is a realisation that it is more important to do the right things than to do things right.

VI

Alfred Korzybski's dictum "The map is not the territory[1]" is especially relevant for those documenting software systems, and those using such documents.

VII

Conventional design education and training (including doctoral research) emphasises one-off solution creation, and pays insufficient attention to reuse, multidiscipline teamwork and the demands and benefits of repetition. This places design education at odds with the vast majority of "real-world" design activities. [private discussion H Wortmann]

VIII

Working in academia and industry provides many advantages; practical experience of the real world to the student, and a break from the operational "fire-fighting" for the worker. Furthermore, it demonstrates that 1 + 1 = 3; also in terms of effort.

IX

The hidden importance of some components in an otherwise well-architected system is illustrated by the impossibility of carrying a cup of tea (while on crutches) when suffering from a knee-injury.

---

[1] In Dutch "De landkaart is niet het land"

X

There is a big difference between evolving and revolving: one is characterised by forward motion towards a better place, the other by repeated circular motion around a fixed point. A parallel can be drawn between organising and re-organising.

XI

While the "what" questions undoubtedly underpin all scientific knowledge, unless the broad research community devotes some (more) time to the "how" questions; the industrial community will continue to prefix "so" to the "what" research, and the research results will remain confined to the research community.

XII

The biggest obstacle to learning a new language is not the ability of the resident population to speak the visitor's language; but the visitor's "pride", which prevents him/her making the mistakes inherent in learning.

XIII

The coexistence of a monarchy and an elected parliament represents a diluted form of democracy – perhaps still acceptable, but not the real thing.

XIV

Een *afspraak maken* om "even *gezellig* een kopje koffie te drinken" is een paradox.

XV

Excessive working on the planning of information-system families hinders planning one's own family.