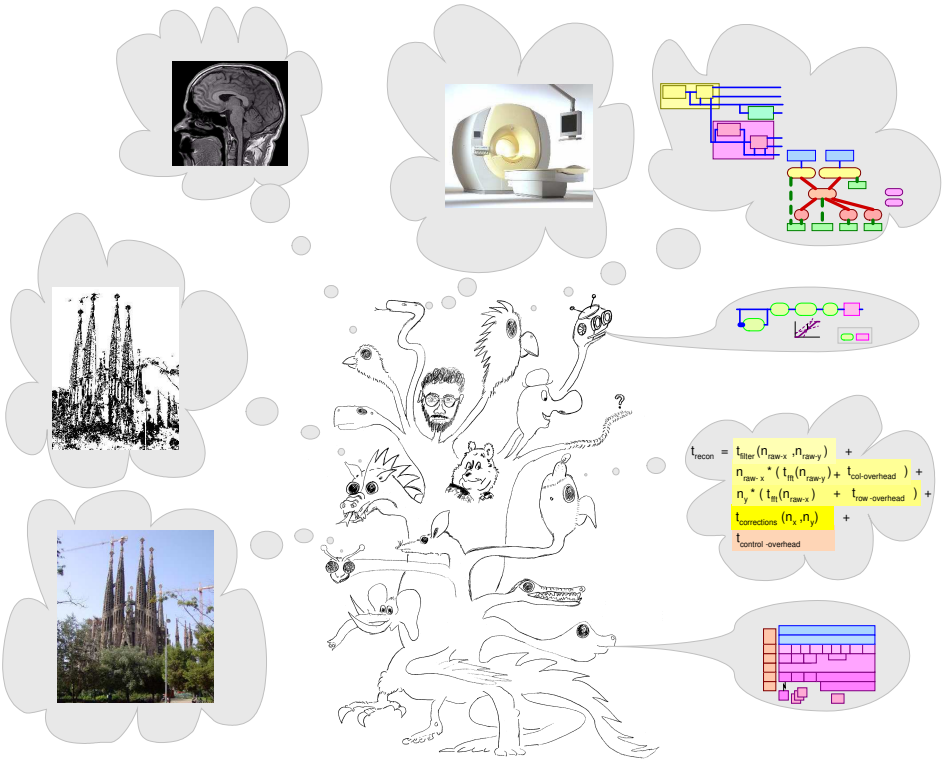
CAFCR: A Multi-view Method for Embedded Systems Architecting;

Balancing Genericity and Specificity



Gerrit Muller

This page will not be present in the final thesis

CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 7 juni 2004 om 13:00 uur
door Gerrit Jan MULLER
doctorandus in de natuurkunde
geboren te Amsterdam

Dit proefschrift is goedgekeurd door de promotor:
Prof.dr W.G. Vree

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr. W.G. Vree	Technische Universiteit Delft, promotor
Prof.dr. M. Rem	Technische Universiteit Eindhoven
Prof.dr.ir. P.A. Kroes	Technische Universiteit Delft
Prof.dr.ir. H.J. Sips	Technische Universiteit Delft
Prof.dr. R. Wagenaar	Technische Universiteit Delft
Prof.dr.ing. D.K. Hammer	Technische Universiteit Eindhoven
Prof.dr. P.H. Hartel	Universiteit Twente

Prof.dr. M. Rem heeft als begeleider in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

ISBN 90-5639-120-2

Keywords: Systems Architecture, System design, Systems Engineering

These investigations were supported by Philips Research Laboratories, and the Embedded Systems Institute, both in Eindhoven.

Cover Photograph: René Stout `http://www.rwstout.com/`

Preface

The industrial world and the academic world have drifted far apart in the immature discipline of systems architecting. The challenge in writing this thesis was to capture the industrial pragmatic experience in a way that is acceptable for the academic community. Moreover, the Embedded Systems Institute has the ambition to mature and extend this discipline. This thesis is a first step in bridging the gap between industry and academics.

The personal motivation for writing this thesis is best explained by looking back at my career. I have been active in the hectic product creation environment for about twenty years. More than 17 years I was responsible for several architectures of medical systems. The drawing on the cover has been made in this period to visualize the system designer as a monster with twenty heads, and hence twenty viewpoints. For more than two years I have been heading the *Systems Engineering* department at ASML. This period at ASML was very instructive and refreshing. I very much liked the opportunity to educate and coach future as well as senior system engineers in this young discipline. After these twenty years of industrial pressure I joined Philips Research, with as my personal goal to mature the discipline and to make it more accessible and transferable. The Gaudí site `www.extra.research.philips.com/natlab/sysarch/` shows the ongoing results of this effort. The step from Philips Research to the Embedded Systems Institute[1] creates the opportunity to link this experience based work to the academic world.

One of the peculiarities of the academic world is that only people with a doctor's title seem to be taken seriously. In the process of writing this thesis I have hit many more aspects of the scientific culture, such as the need to cite others for every statement being made, an extremely redundant writing style, the preference for text over figures, and a focus on the argumentation rather than on the clarity or didactic value. Going through this learning experience is very valuable in

[1] The Embedded Systems Institute is founded by Philips, ASML, Océ, TNO and the universities of Delft, Eindhoven and Twente

the interaction with the academic world. I gained a better understanding of the obsession for publication, peer review, and the noise induced by all citations and repetitions in scientific publications. Understanding the motivation and the behavior of the academic partners is a prerequisite to bridge the gap between industry and academics.

My thinking processes make use of visualizations: diagrams, structures, models, et cetera. This thesis contains over 150 figures. The advantage of figures is that they show the overview and the parts at the same time, whereas text is linear: only after reading the entire text the overview might become clear. Lots of energy has been spent to complement all visualizations with readable text.

The broad and multi-disciplinary scope of systems architecting can easily result in generic statements. I have attempted to cope with this danger of overgeneralization by providing a very specific case study. This struggle with the balance between genericity and specificity is the daily world of every systems architect. This struggle repeated itself when writing this thesis. The academic value is in the extraction of more generally applicable knowledge. But the route towards this knowledge meanders through many highly specific details.

If both academic people as well as industrial people enjoy reading this thesis then one of my goals has been achieved. So, enjoy reading!

Contents

Introduction

This thesis describes an architecting method that is intended to help architects in creating embedded systems. The effort to create embedded systems is increasing exponentially. At the same time system complexity increases significantly, threatening performance, reliability, and other system characteristics. The architecting method described here is intended to help the architect to cope with the ever increasing complexity. The method described integrates the *"CAFCR" model* (Section 2.3, and Chapters 8 and 9), *design via qualities* (Chapter 10), *story telling* (Chapter 11), and *threads of reasoning* (Chapter 12) into an open-ended architecting method (Part II). The method is based on reflection of 20 years of creating complex *software and technology intensive* systems.

A lot of effort is spent in determining an approach to do research on architecting methods in an accountable way, described in Chapter 5. The architecting method is mapped on a case of product creation in an industrial environment: a Medical Imaging Workstation (Part III). This case is used to evaluate the architecting method (Chapter 18). Additional evaluation information is provided in the context of research projects, workshops and courses (Chapter 19).

Figure 1 shows the structure of this thesis. Part I provides the context and goal, provides a preview of the method, introduces the case positions the method, and describes the research method. The theoretical framework of the architecting method is described in part II. Part III describes the case: medical imaging workstation. Part IV evaluates the architecting method and discusses the balance between the need for *abstraction*, resulting in genericity, and the need for *details*, requiring to be specific.

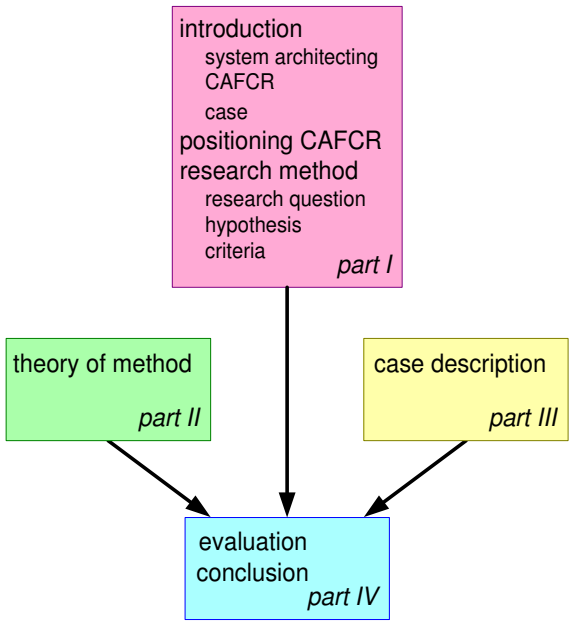


Figure 1: Structure of this thesis

Recommended literature and other resources:

- "The Art of Systems Architecting", Rechtin [68]
- "Systems Engineering Guidebook", Martin [46]
- "Resources for Software Architects", Bredemeyer [11]
- "Role of the Software Architect", Bredemeyer [12]

This thesis focuses on the integration aspects of the methods, with the CAFCR model as core. The wider systems architecting context, the method itself and the supporting submethods are more extensively described at the Gaudí site: `http://www.extra.research.philips.com/natlab/sysarch`. This thesis and more supporting articles can be found at: `http://www.extra.research.philips.com/natlab/sysarch/ArchitecturalReasoning.html`.

Part I

Introduction to CAFCR and Threads of Reasoning

Chapters in Part I:

Chapter 1

What is Systems Architecting in an Industrial Context?

1.1 Introduction

This thesis discusses the systems architecting of software and technology intensive products. Typical examples of software and technology intensive products are televisions, DVD-players, MRI scanners, and printers. The creation of these products is a multi-disciplinary effort by hundreds of engineers. The time between first product ideas and introduction into the market is in the order of a few months to a few years.

The concept *architecture* is borrowed from the building discipline. *Architecture* in building has a long history, with well known names as Vetruvius, Gaudí , Lloyd Wright, Koolhaas, and many many more. *System architecture* can be compared with building architecture. The architecture of a building is for a large part the experience that people get when they interact with the building, ranging from "how does it fit in the environment?", "what impression does it make?", "is it nice to be there?", to "is it useful?". In other words, the less tangible aspects of the perception of the building and the experience with the building are important aspects of the architecture. The technical aspects of the structure and the construction of the building are also part of the architecture. The feasibility of an architectural vision is enhanced or constrained by these technical aspects. The architecture is a dynamic entity that evolves during the life-cycle of the building. Every phase has its own particular needs. Early-on the constructibility is important; later the usability and adaptability, and finally the disposability, become the points of at-

tention.

In this book the system architecture is a close metaphor of the building architecture. The system architecture covers both the external aspects, often intangible such as perception and experience, and the internal aspects, often more tangible such as structure and construction. Note that this definition of architecture is rather broad, much broader for instance than usual in the software architecture community, see the Software Engineering Institute (SEI) inventory [37] for a much wider variation of definitions for architecture. Essential in this definition is the inclusion of the user context in architecture.

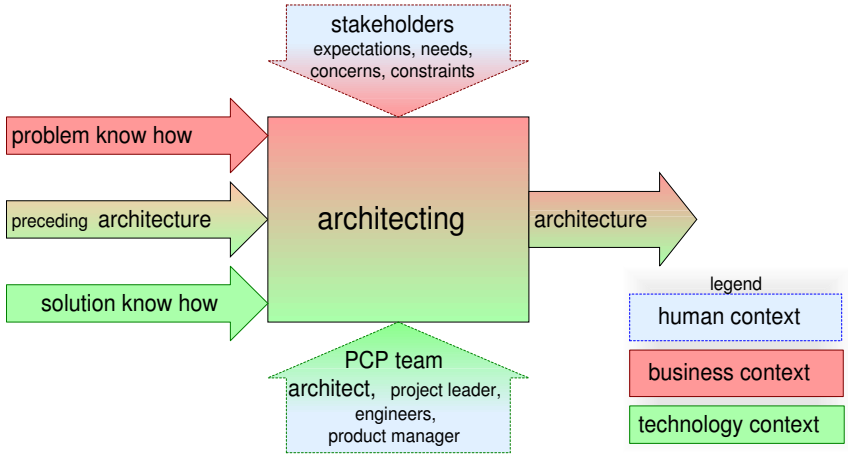


Figure 1.1: Architecting = creating an architecture

The activity of creating an architecture is called architecting, see Figure 1.1. The process of creating a new product is called Product Creation Process (PCP). A multi-disciplinary team, the PCP team, creates the product. The input to the PCP comes from all stakeholders, with their needs, concerns, expectations, et cetera. The architect is responsible for the quality of the architecture: a system that meets the stakeholder's expectations, that provides the stakeholders with an attractive and useful experience, and that can be realized by the PCP team.

The architecting activity transforms problem and solution know how into a new architecture. In most cases the architecting is done by adapting preceding architectures. The preceding architecture is an input for the architecting effort. Green field architectures (problems without existing architecture, or where the existing architecture can be completely ignored) are extremely rare.

1.2 Description of the Business Context

Architecting methods are positioned in the business context by means of a variant of the “BAPO”-model [58]. The business objectives of the company are the main inputs for architecting: generating market share, profit, ratio between sales and investments, et cetera. The specific business objectives depend strongly on the domain: the type of product, customers, competition, application and market.

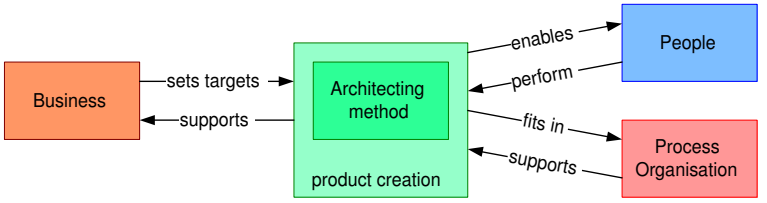


Figure 1.2: The business context of architecting methods

The business context is shown in Figure 1.2. The business will set targets for the architecting methods, the architecting methods will support the business. The product creation uses an architecting method to develop new products. The architecting method must fit in the processes and the organization. People do the real work, the method should help people to architect the desired system.

1.3 Internal Stakeholders

Many stakeholders in the business context are involved in the creation, production, sales and service of the products. All these operational stakeholders have their own concerns. These concerns translate into needs that influence the product specification. Figure 1.3 shows the internal stakeholders as annotation to figure 1.2.

The *policy and planning process* sets the strategy and anticipates on the longer term future. The scope of this process is at portfolio level. The *policy and planning process* has the overview and strategic insight to allow decisions about product synergy and optimizations across products and product families. Also decisions about involving partners and the degree of outsourcing are taken here. These internal strategic considerations also translate into operational requirements.

The *customer-oriented process* covers the entire order realization process as well as the sales and life-cycle support (service) processes. Manufacturability, serviceability, and many more requirements are determined by these stakeholders.

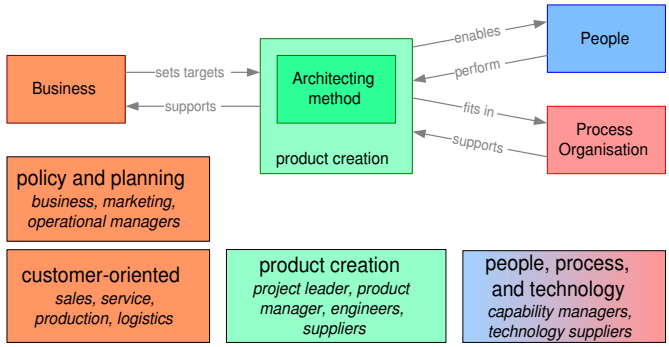


Figure 1.3: Stakeholders of the product creation within a company itself

All specification and design work is done in the *product creation process*. Many contacts with internal and external suppliers take place during product creation. The operational needs of this process, such as work breakdown, test models, et cetera, also result in operational requirements.

The *people, process, and technology management* is concerned with processes, methods, tools, skills of people, intellectual property, and technology development. These concerns will sometimes result in operational requirements. Care should be taken that the justification of these requirements is clear. From a business point of view these issues are means that must serve the business goals, not the other way around.

1.4 Acknowledgements

Richard George attended me on the correct spelling of *Lloyd Wright*.

Chapter 2

Overview of CAFCR and Threads of Reasoning

2.1 Introduction

At the beginning of the creation of a new product the problem is often ill-defined and only some ideas exist about potential solutions. The architecting effort must change this situation in the course of the project into a well articulated and structured understanding of both the problem and its potential solutions. Figure 2.1 shows that basic methods and an architecting method enable this architecting effort.

The basic methods are methods that are found in a wide range of disciplines, for example to analyze, to communicate, and to solve problems. These basic methods are discussed in Chapter 7.

An overview of the architecting method is given in Section 2.2. The architecting method contains multiple elements: a framework, briefly introduced in Section 2.3, and submethods and integrating methods, which are described in part II.

2.2 Architecting Method Overview

Figure 2.2 shows the overall outline of the architecting method. The right hand side shows the visualization as it will be used in the later chapters. The *framework* is a decomposition into five views, the "CAFCR" model, see Section 2.3.

Per view in the decomposition a collection of *submethods* is given. The collections of submethods are open-ended. The collection is filled by borrowing relevant

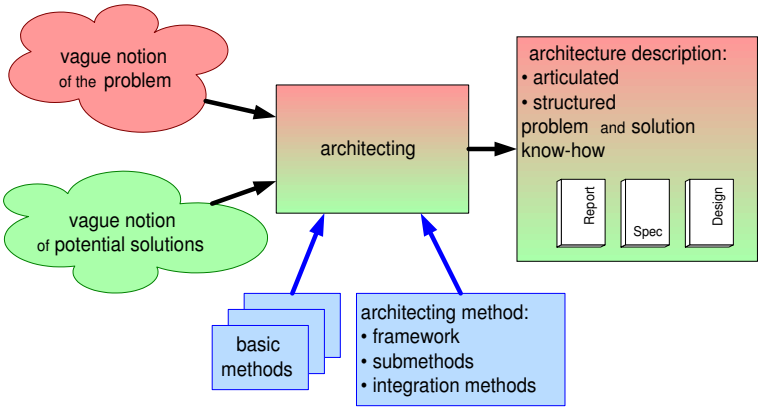


Figure 2.1: An architecting method supports the architect in his process to go from a vague notion of the problem and a vague notion of the potential solutions to a well articulated and structured architecture description

methods from many disciplines.

A decomposition in itself is not useful without the complementing integration. *Qualities* are used as *integrating* elements. The decomposition into qualities is orthogonal to the "CAFCR" model.

The decomposition into CAFCR views and into qualities both tend to be rather *abstract*, *high level* or *generic*. Therefore, a complementary approach is added to *explore specific details*: story telling. Story telling is the starting point for specific case analysis and design studies.

These approaches are combined into a thread of *reasoning*: valuable insights in the different views in relation to each other. The basic working methods of the architect and the decompositions should help the architect to maintain the overview and to prevent drowning in the tremendous amount of data and relationships. The stories and detailed case and design studies should help to keep the insights factual.

2.3 The CAFCR Model

The "CAFCR" model is a decomposition of an architecture description into five views, as shown in Figure 2.3. The *customer objectives* view (**what** does the customer want to achieve) and the *application* view (**how** does the customer realize his goals) capture the needs of the customer. The needs of the customer (**what**

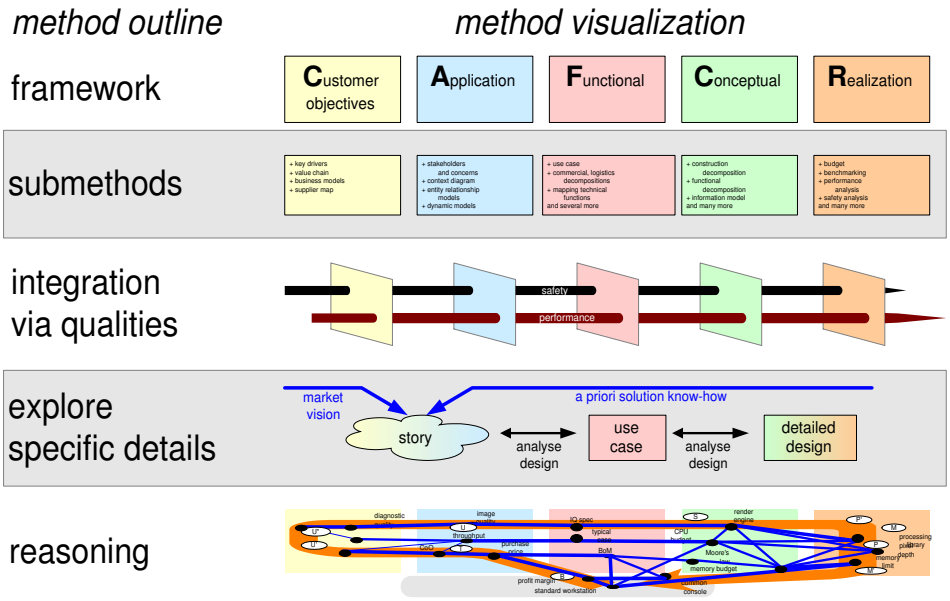


Figure 2.2: The outline of the architecting method with the corresponding visualization that will be used in the later chapters.

and **how**) provide the justification (**why**) for the specification and the design.

The *functional* view describes the **what** of the product, which includes (despite its name) the *non-functional* requirements.

The **how** of the product is described in the *conceptual* and *realization* views. The **how** of the product is split into two separate views for reasons of stability: the conceptual view is maintained over a longer time period than the fast changing realization (Moore's law!).

The job of the architect is to integrate these views in a consistent and balanced way, in order to get a *valuable*, *usable* and *feasible* product. Architects do this job by continuously iterating over many different viewpoints, sampling the problem and solution space in order to build up an understanding of the business. This is a top-down approach (objective driven, based on intention and context understanding) in combination with a bottom-up approach (constraint aware, identifying opportunities, know-how based), see Figure 2.4.

The CAFCR model in Figure 2.4 is focused on the relation between the customer world and the product. Another dimension that plays a role in specification

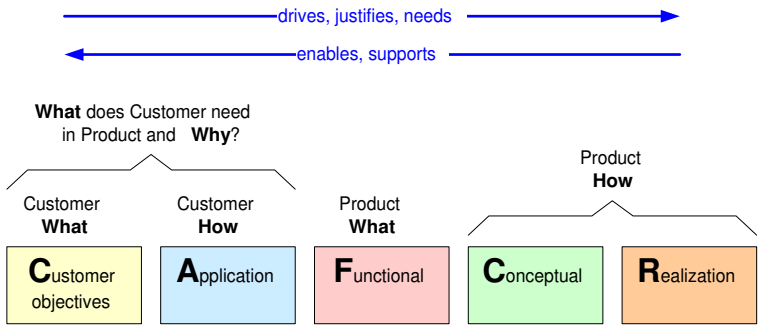


Figure 2.3: The "CAFCR" model

and design is the *operational* view. The operational view describes the internal requirements of the company: what is needed for the operation of the company? The CAFCR model is focused on the customer world: what determines *value* and *usability* of a product? The business *feasibility* of a product is largely determined by the operation of the company: satisfactory margins, service levels, potential for the future. Strategic requirements of the company, which are important for the long term operation, are also part of the *operational* view.

The customer views and operational view are asymmetric. The customer world is outside the scope of control of the company. Customers have a free will, but act in a complex environment with legislation, culture, competition, and their own customers, who determine their freedom of choices. The operational way of working of a company is inside the scope of control of the company. The company is also constrained by many external factors. Within these constraints, however, the company decides itself how and where to manufacture, to sell, and to provide service. The operation of the company is organized in such a way that it supports its customers. The asymmetry is that a company will never tell its customers to organize in a way that eases the operation of the company[1]. The operational view is subject to the customer views.

The CAFCR views and the operational view must be used concurrently, not top down as in the waterfall model. However, at the end of the architecting job a consistent description must be available, see [62]. The *justification* and the *needs* are expressed in the Customer Objectives View, the Application View, and the op-

[1]In practice it is less black and white. A company interacts with its customers to find a mutual beneficial way of working. Nevertheless, the provider-customer relationship is asymmetric. If the provider dictates the way of working of the customer then something unhealthy is happening. Examples of unhealthy relations can be found in companies with a monopoly position.

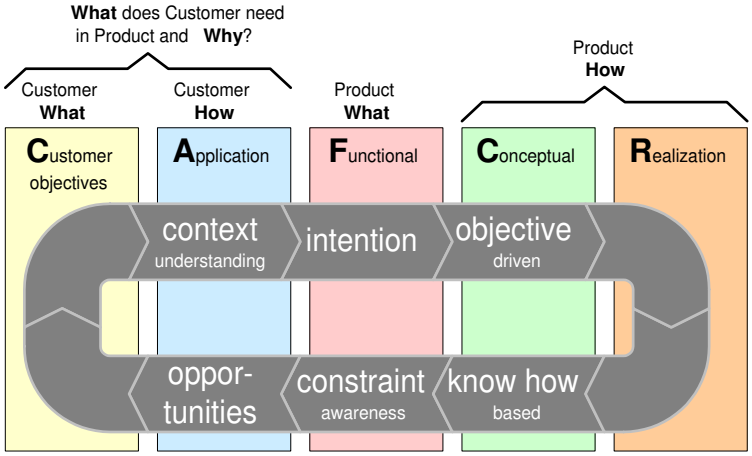


Figure 2.4: Iteration over the CAFCR views and the operational view. The task of the architect is to integrate all these viewpoints, in order to get a *valuable*, *usable* and *feasible* product.

erational view. The technical solution as expressed in the Conceptual View and the Realization View *supports* the customer to achieve his objectives and support the company in the operation. The Functional View is the interface between problem and solution world.

The CAFCR model will be used in this thesis as a framework for a next level of submethods. Although the five views are presented here as sharp disjunct views, many subsequent models and methods don't fit entirely into one single view. This in itself is not a problem; the model is a means to build up understanding, it is not a goal in itself.

The "CAFCR" model can be used recursively: many customers are part of a longer value chain and deliver products to customers themselves. Understanding of the customer's customer improves the understanding of the requirements.

The notion of **the** customer is misleading. Many products have an extensive set of stakeholders in the customer domain. One category of customer stakeholders are decision makers such as: CEO (Chief Executive Officer), CFO (Chief Financial Officer), CIO (Chief Information Officer), CMO (Chief Marketing Officer) and CTO (Chief Technology Officer). Another category are people actually operating the system, such as users, operators, and maintainers. A last category mentioned here are the more remotely involved stakeholders, such as department chiefs and purchasers.

Chapter 3

Introduction to Medical Imaging Case Study

3.1 Market and Application

The Easyvision is a medical imaging workstation that provides additional printing functionality to URF X-ray systems, see Figure 3.1. In a radiology department three URF examination rooms can be connected to a single Easyvision workstation. The Easyvision can process and print the images of all three URF systems on transparent film. The radiologist is viewing the film on a light box to perform the diagnosis.

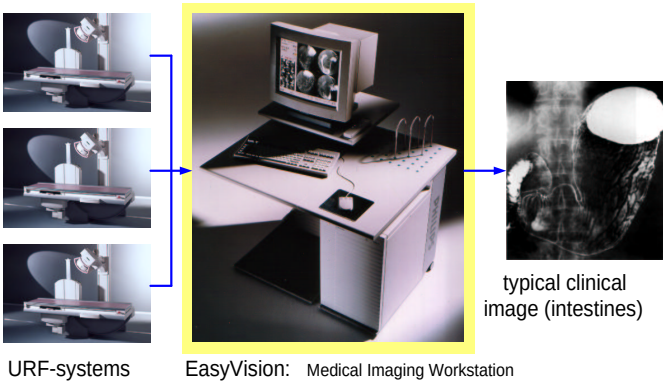


Figure 3.1: Easyvision serving three URF examination rooms

URF systems are used in gastrointestinal examinations. The patient has to consume barium meal to enhance the contrast. Multiple exposures are made at different locations in the intestines, while the barium meal progresses. The radiologist applies wedges to expose the area of interest and to minimize the X-ray dose for the rest of the body.

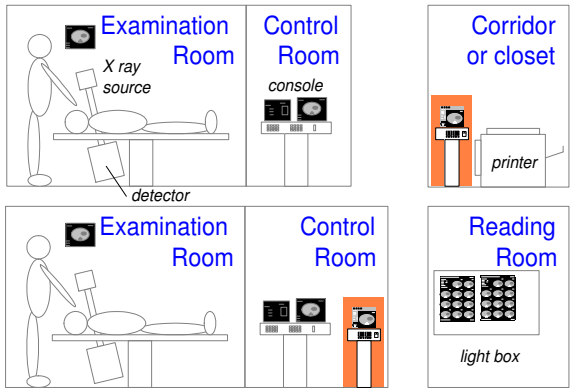


Figure 3.2: X-ray rooms with Easyvision applied as printserver

Around 1990 the normal production of transparent film was performed by means of a multi-format camera that makes screen copies of the CRT-monitor. The operator selects every image and sends it to the camera. A typical radiology department layout is shown in Figure 3.2.

The introduction of the Easyvision made it possible to connect three examination rooms via an Easyvision to a digital laserprinter. Figure 3.2 shows that the Easyvision can be positioned as a server in some cabinet, in which case the system is used remotely, without any direct operator interaction. The Easyvision can also be placed in one of the control rooms, thereby enabling manual processing of the images and manual formatting of the film.

The introduction of an Easyvision can immediately be justified by reduced film costs. Figure 3.3 shows a comparison of the conventional way of working, where images are screen copies of the CRT-monitor, and the films obtained by means of software formatting, where the film layout can be optimized to maximize the number of images.

The conventional way of working results in many duplicates of the textual information around the image itself, because for each image the complete screen is copied. This is a waste of film space. On top of that all the textual information is high contrast information, which is distracting while viewing for the diagnosis.

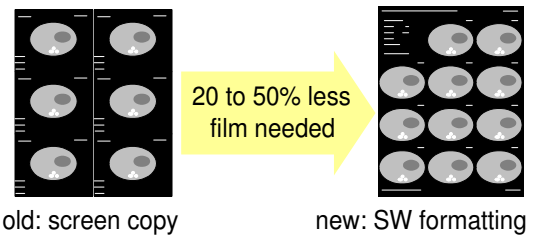


Figure 3.3: Comparison screen copy versus optimized film

The digital availability of images opens all kinds of possibilities. The simplest is the separation of duplicate text information and images, which makes a much higher packing of images possible. Secondary possibilities are automatic shutter detection and zoom-to-shutter.

3.2 Technology

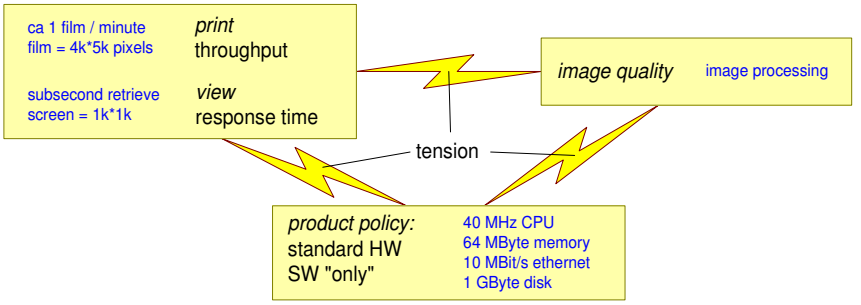


Figure 3.4: Challenges for product creation

The vision of the original designers of the product was that the technological innovation in computer hardware is so fast that proprietary hardware development would hamper future product innovation. A product policy was chosen to create products with the value in the software, using standard off-the-shelf hardware. This policy is potentially in conflict with the performance and image quality requirements. This challenge is shown and annotated in Figure 3.4.

Two types of performance are important in this product: throughput (the amount of film sheets printed per hour) and response time (the user interface re-

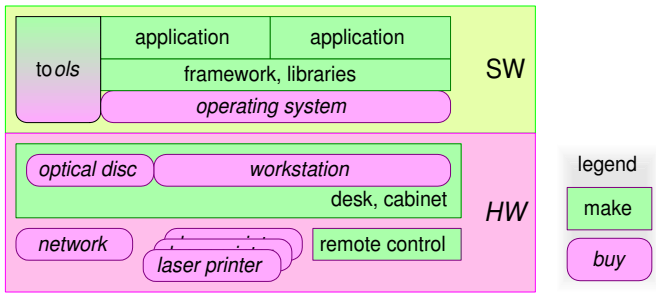


Figure 3.5: Top-level decomposition

sponse time should be subsecond for image retrieval). This performance must be achieved with a minimal guarantee in image quality. For instance, pixel replication for still images on screen is not acceptable, while bi-cubic interpolation is required for the high resolution of the film images. These requirements must be realized with the workstation in the 5 to 10 k$ range of that time, which corresponds with a 40 MHz CPU and a maximum amount of memory of 64 MByte. The examination rooms are connected to the system via 10 Mbit ethernet, which was state of the art in 1990.

Figure 3.5 shows the top-level decomposition of the system. Most hardware is off-the-shelf. A custom remote control was added to obtain a very direct and intuitive user interface. In order to fit the system in the hospital environment, the packaging of the system was also customized. The packaging part of the system was decoupled from the hardware innovation rate by a box in a box concept: the off-the-shelf computer box was mounted in a larger deskside-cabinet.

The software is based on a standard operating system (Unix), but the libraries, framework and applications are tailor-made. The framework and libraries contain a lot of clinical added value, but the end user value is in the applications.

The designers of Easyvision introduced many technological innovations in a relatively conservative product creation environment. The following list shows the technological innovations introduced in the Easyvision:

- standard UNIX-based workstation
- full SW implementation, more flexible
- object-oriented design and implementation (Objective-C)
- graphical User Interface, with windows, mouse et cetera
- call back scheduling, fine-grained notification

- data base engine: fast, reliable and robust
- extensive set of toolboxes
- property-based configuration
- multiple coordinate spaces

The introduction of these innovations enabled the later successful expansion into a family of products, with many application innovations. In Part III we will show some of these innovations in more detail and in relation to the product value.

Chapter 4

Positioning the CAFCR Method in the World

4.1 Introduction

This chapter positions the "architectural reasoning" *architecting method* relative to other engineering and architecting methods.

Section 4.2 describes work that is related to the research of architecting methods. Section 4.3 articulates explicitly the specific contribution of this thesis. The IEEE 1471 is explained further in section 4.4, because its contents is highly relevant in this context.

4.2 Related Work

Conventional disciplines, such as mechanical engineering, electronic engineering, et cetera have a clear set of methods and tools. Students can learn the discipline by attending universities and following their curriculums.

This is not the case for systems architecting. Only a few universities teach systems architecting. There are multiple reasons for the fact that teaching systems architecting methods at universities is difficult. First of all, sufficient depth of engineering know-how is needed to be able to work in the architecting area. In other words, a conventional discipline is a prerequisite to become an architect.

Secondly, architecting is done for problems with a wider scope than conventional engineering problems. The larger the scope, the more ill-defined a problem

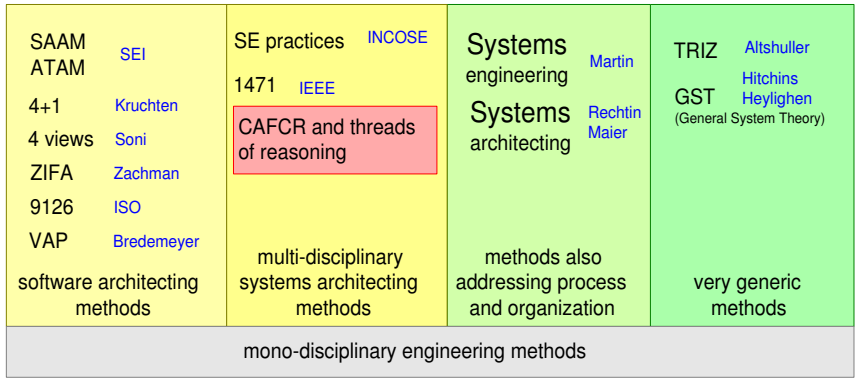


Figure 4.1: Classification of architecting methods

becomes. The methods range from *flexible* for ill-defined problems to *rigid* for well-defined problems[1].

Figure 4.1 shows a classification of architecting methods, with the scope of the method as differentiating factor. The software architecting methods have the smallest scope. System architecting methods widen the scope to system level. This thesis addresses the multi-disciplinary systems architecting methods. The scope can be further increased to include processes and organizational issues. The widest scope pertains to very generic methods, which claim to be domain agnostic and to create value by cross-fertilization across domains. At the bottom of the classification we find the mono-disciplinary methods, which are the fundamentals on which all methods build.

4.2.1 Software Architecting Methods

A whole class of methods originate in the Information Technology (IT) world and address software architecting. The software architecting methods do not address the system level problems, such as hardware/software trade-offs.

The Software Engineering Institute at Carnegie Mellon University, [71] and [72], increases the problem scope and puts a lot of emphasis on processes, and restricts itself to software architecture. Examples of methods developed here are Software

[1]Of course this is an oversimplification. Sometimes agile methods are highly effective in well-defined problems. Sometimes rigid methods can perform wonders in an ill-defined problem. In general, mature methods are available for well-defined problems, while the uncertainty in ill-defined methods requires more flexibility.

Architecture Analysis Method (SAAM) [41] and Architecture Trade Off Analysis Method (ATAM) [40].

Zachman provides a framework for enterprise architectures, see [80]. This framework defines two dimensions with six aspects each, creating a space with 36 different views. Bredemeyer describes a nice visual method “The Visual Architecting Process” [13]. The Bredemeyer method provides context views and a path from context views to design views. Both Zachman and Bredemeyer are software oriented.

Well known multi-view software architecting methods are Soni [33], and the 4+1 method from Kruchten [43]. These two methods use multiple views. The scope of Soni methods, however, is completely limited to the technical solution domain. Kruchten is also focused on the technical solution domain, but he makes a small step into the problem domain by use cases in the fifth view.

ISO 9126 [38] is a standard that consolidates a quality framework. The framework addresses the same type of qualities that are discussed in chapter 10. Unfortunately ISO 9126 limits itself to software only.

4.2.2 Multi-disciplinary System Architecting Methods

A further increase in scope can be found in the *Systems Engineering Community*, with INCOSE[36] (International Council on Systems Engineering) as representative organization. All stakeholders are taken into account and the full life-cycle is emphasized. Examples of this approach can be found at the INCOSE web site [21].

Some standardization work has been done in the scope of systems, stakeholders and the full life cycle. An example is IEEE 1471, which is a framework that fits into this scope, see section 4.4.

This thesis about architectural reasoning, based on the “CAFCR” method, also addresses the scope of systems, their stakeholders, and the full life-cycle. Boundary conditions to the methods in this thesis are structure and characteristics of the business, the organizations, and the processes.

4.2.3 Methods also Addressing Process and Organization

The architect is often confronted with many more needs, worries, and complications, originating from human and business aspects. This broad working environment is full of uncertainties. Rechtin and Maier [68] address this wider scope from the architecting point of view. Martin [46] comes from the systems engineering

community. He provides a method that deals with all the complexity, but that has less emphasis on the human aspects.

4.2.4 Very Generic Methods

Many system architecting and design methods are universally applicable. General Systems Theory (GST), for example, addresses any kind of system, ranging from economical, or ecological, to social, see for instance [31] and [32]. GST suffers from being extremely abstract and difficult to apply, due to a broad scope and the generic nature of the theory.

TRIZ [1] is a methodology for innovation that originates in Russia. A set of innovation patterns is derived from studying large collections of inventions. These patterns are transformed into innovation methods that can be applied to a very broad range of applications. One of the starting points of TRIZ is that the way of innovating in one domain provides inspiration for innovation in other domains. TRIZ provides a number of useful insights.

The subtitle of this thesis, *balancing genericity and specificity*, indicates one of the continuous struggles of the architect: the power and the beauty of generic solutions versus the uniqueness of effective, individual solutions. Or in other words, do we get carried away in generic thinking, or do we drown in the details? In this thesis the scope will be limited to systems with embedded processors and software. This still pertains to a very broad range of products: from wafersteppers to televisions, and to systems on a chip).

4.3 What is the Unique Contribution of this Work?

This section discusses the unique contributions of the CAFCR method. Although every single element mentioned here is present in one of the discussed methods, the uniqueness of CAFCR is the combined application of all these elements simultaneously.

Integral and Multi-disciplinary This work focuses on architecting methods on the *system* level for embedded systems. As described in Subsection 4.2.1, many methods focus only on a part of the multi-disciplinary system problem, for instance only on the software architecture. A lot of architecting methods provide more or less closed and complete solutions. The available methods are partial methods from a systems viewpoint. The method described in this thesis addresses the integration of results obtained with these

more partial methods. Also a number of multi-disciplinary system design submethods are described in this thesis. The basis for this integration is the combined use of CAFCR views, qualities, and threads of reasoning.

Goal-Oriented This method stresses the importance of being externally oriented. Architecting must be goal-oriented or objective-driven. Many existing methods do not take the goals and objectives into account.

Practical, based on Industrial Experience The method, which is based on a broad industrial experience, addresses the real problems[2] in system design. The usability aspect can be seen in the light-weight use of formulas, and in the association of many statements with common sense. Some of the published methods are more academic, well thought through, but not really addressing the problems in system design, and difficult to implement in the industrial practice.

Flexible The wide application range of the creation of *software and technology intensive products*, requires a flexible and adaptive method. The method must provide guidance, and should not constrain the architect by forcing a rigid harness on him. In principal the *architecting method* must be able to integrate the results of *any* partial method.

Builds on standards The method builds on top of standards, such as ISO 9126 for qualities and IEEE 1471. In fact the method can be viewed as an instantiation of an IEEE 1471 method, see Section 4.4.

Support for short innovation cycles System engineering methods originate from the aerospace domain, with very different reliability and safety requirements. Such methods tend to be more rigid, resulting in very long development cycles. This distinction of “slow but safe” domains versus “fast but less reliable” domains disappears quickly. Cross-fertilization of these domains can be very useful. In contrast to the aerospace domain the *CAFCR* method is intended for domains with short innovation cycles.

[2]Many problems in system design are caused by unforeseen interactions between independent designed functions or qualities. See for instance Chapter 13 for examples of system design problems in the Medical Imaging case.

4.4 IEEE 1471

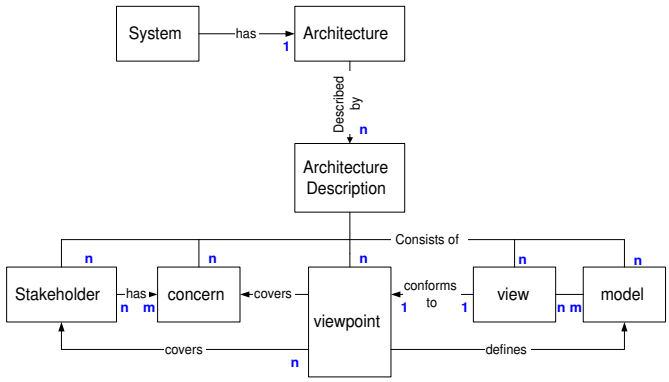


Figure 4.2: The IEEE 1471 model for stakeholders, viewpoints and architecture descriptions

Figure 4.2 shows a somewhat simplified IEEE 1471 model. IEEE 1471 [6] is a standard that describes a framework for architecting. The framework introduces a number of important concepts:

Stakeholders People or organizations that have an interest in the system under consideration.

Concerns The articulation of the needs and worries of the stakeholders.

Viewpoints The points of view used to describe part of the problem or solution. IEEE 1471 makes a subtle difference between *view* and *viewpoint*. We ignore this difference here.

Models Frequently used method to make problem and solution descriptions.

Architecture description The combination of stakeholders, concerns, viewpoints and models to describe the architecture of a system.

The main contribution of IEEE 1471 is to provide a framework that covers all of these aspects. The individual concepts have been in use by many architects for a long time.

On top of providing the framework, IEEE 1471 also recognizes the fact that complete consistency in the entire architectural description is an illusion. The

real world of designing complex systems is full of stakeholders with fuzzy needs, often contradictory in itself and conflicting with needs of other stakeholders. The insights of individual designers are also full of different and changing insights. This notion of incomplete consistency is not an excuse for sloppy design; quite the opposite: recognizing the existence of inconsistencies is a much better starting point for dealing with them. In the end, no important inconsistencies may be left in the architecture description.

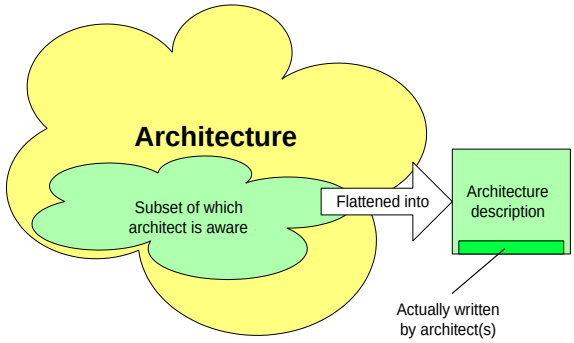


Figure 4.3: The architecture description is by definition a flattened and poor representation of an actual architecture.

IEEE 1471 makes another interesting step: it discusses the architecture *description* not the *architecture* itself. The *architecture* is used here for the way the system is experienced and perceived by the stakeholders[3].

This separation of *architecture* and *architecture description* provides an interesting insight. The *architecture* is infinite, rich and intangible, denoted by a cloud in figure 4.3. The *architecture description*, on the other hand, is the projection, and the extraction of this rich *architecture* into a flattened, poor, but tangible description. Such a description is highly useful to communicate, discuss, decide, verify, et cetera. We should, however, always keep in mind that the description is only a poor approximation of the *architecture* itself.

[3]Long philosophical discussions can be held about the definition of **the** architecture. These discussions tend to be more entertaining than effective. Many definitions and discussions about the definition can be found, for instance in [32], [10], or [37]

Chapter 5

Research in Systems Architecting

5.1 Introduction

Architecting is an extremely broad subject, taking into account many ill-defined needs, concerns, expectations, et cetera. *Architecting methods* are the result of consolidation of experience of architects. *Architecting methods* should help architects to architect. *Research of architecting methods* is again one step more abstract: it is the study and exploration of *architecting methods* in a systematic way.

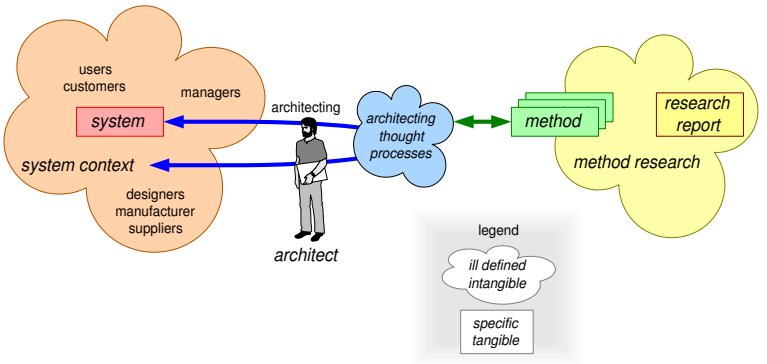


Figure 5.1: Research of architecting methods in the context of system design

Section 5.2 describes a *technology management model*. The *technology management model* is used in Section 5.4 to describe the research method used in this thesis to investigate an architecting method. Section 5.3 discusses the chal-

lenges of doing research of architecting methods in a scientific way. Section 5.5 describes how the distance between industrial practice and scientific research can be bridged. Section 5.6 describes the environment where the research takes place.

5.2 Technology Management Cycle

The creation of software and technology intensive products requires by definition quite some technology know-how. These technologies can be classified as *hard* and *soft*:

- *Hard* technology is the tangible engineering and scientific know-how, such as software and electronics engineering, and mathematics, physics, chemistry, and biology. The know-how from these sciences is very objective and universally applicable (the elasticity in the USA is the same as the elasticity in China). The performance of the *product* is determined by the right choice of *hard* technologies.
- *Soft* technology is the less tangible know-how of how to create a product with a team of people. Soft technologies are based on a mixture of sciences and human arts. The know-how of soft technologies is more subjective, the human factors are less well reproducible (a method working well in the USA might fail in China and vice versa). The performance of the *product creation team* depends on the right application of *soft* technologies.

The intensive use of technology in these products requires explicit management of the technology: technology management. Architecting methods are managed as part of the (soft) technology.

Technology management can be modeled as a cyclic process [17], as shown in Figure 5.2. Most of the time is spent in the application of technology, in other words in the creation of new systems. After applying the technology it is recommended to learn from this application by reflection. The learning experience can be made (partially) accessible to others by consolidating the know-how, for instance in documentation.

At the end of the consolidation insight will exist in strengths and weaknesses of the technology, both in the hard technology choices as well as in the soft technology (the approach taken). It is recommended to take this know-how as a starting point for an exploration phase.

The exploration phase should be used to refresh the designers and architects, and to open new opportunities in technology. This requires that they know the

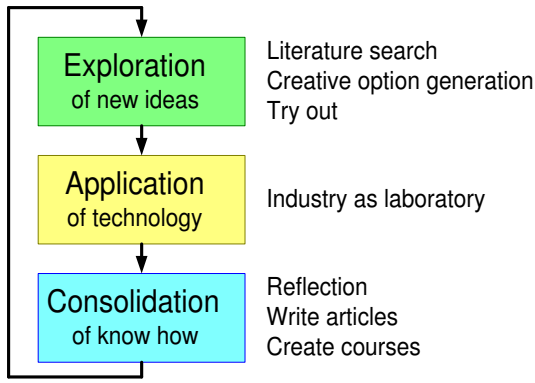


Figure 5.2: Technology Management Cycle

state of the art in the world, by reading literature, visiting conferences, et cetera. New technology options can be added by means of creative brainstorms . Promising technology must be explored hands-on.

In the next application phase a limited set of new technologies is applied in practice.

This thesis focuses entirely on an architecting method. The architect and the architecture are heavily involved with a lot of hard technology. However, the management of hard technologies and soft technologies other than architecting methods is outside the scope of this thesis.

5.3 Challenges to do Research in a Scientific Way

Science is applied in a wide range of areas, from proof-based mathematics to descriptive reasoning in human sciences, see Figure 5.3.

The level of certainty of the results decreases when moving from hard sciences to soft sciences. Mathematical proofs provide certainty[1], see also [35]. Physics provides a confidence level that increases by validating predicted outcomes, or it applies a *falsification* process as described by Popper [76].

Medical sciences need a lot more trial and error, where evidence is built up

[1] As far as the proof is verifiable and the verifiers can be trusted. The absolute certainty is here also decreased by the human factor: the proof is as certain as the quality of the provider of the proof and the verifiers of the proof. Automation shifts the problem to the tool, which also in some way originates in fallible human beings.

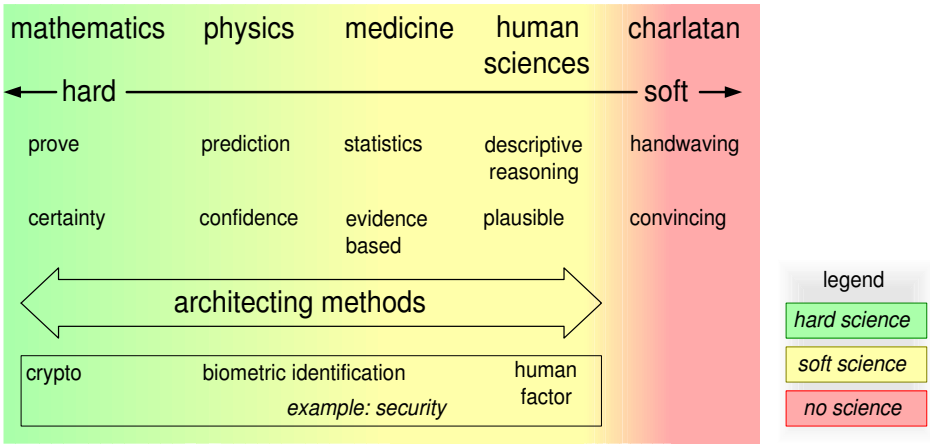


Figure 5.3: Spectrum of sciences

in extensive statistical studies. The evidence is hampered by many factors that influence the outcome of the medical study, but that are outside the control of the experimenter. Worse is that many of the factors are unknown to the experimenter and his peers. Cause and result are often more ambiguous than people realize. Despite all these disclaimers the medical sciences have created a large body of knowledge.

The human sciences (psychology, sociology, pedagogy, et cetera) have already a tremendous challenge in making statements plausible. Human behavior shows a wide variation, depending on many factors, such as culture, age, gender, and status. Individual human behavior is often poorly predictable. Case descriptions are used in a heuristic approach. The step from case descriptions to a workable hypothesis needs a lot of interpretation. Adding more case descriptions will help in making the issue more plausible, but hard evidence is nearly impossible. A more experimental approach with small scale experiments is possible, but these experiments are often highly artificial.

The scientific community dislikes the charlatans, who can be very convincing by hand-waving arguments, but in fact are selling hot air.

Architecting integrates all of these different types of sciences, from mathematical to human sciences. For instance in security design cryptographic proof is important, and also biometrics authentication. However a security solution that does not take the human behavior into account fails even before it is implemented.

Research of architecting methods is inherently the combination of hard facts

in an environment full of soft factors. Most of present-day hard disciplines (mathematics, physics, electronics, mechanics, et cetera) are frightened away by the soft factors. Most of the soft disciplines (psychology, philosophy, business management) have no affinity with the complexity in the hard facts. The challenge in the systems discipline is to tackle the soft factors, with sufficient understanding of the hard side.

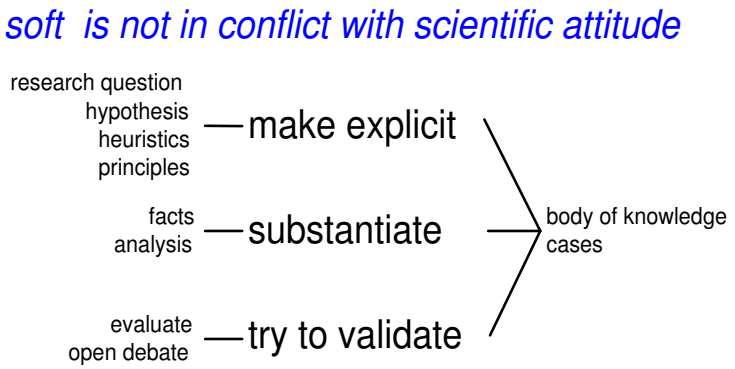


Figure 5.4: Soft problems can be approached with a scientific attitude

The fact that so many soft factors play a role is no excuse to stay in "trial and error" mode. The scientific attitude, see Figure 5.4, can also be applied to the soft kind of problems encountered in systems architecting. The Philosophy of Science has a long history. Some inspiration for the approach taken here are the *falsification* process by Popper, summarized by Tuten in [76], and the notion of *paradigms* by Kuhn, also summarized by Tuten in [77]. Popper formulated the foundation of scientific methodology, for instance based upon open discussion, testable statements and a critical attitude. The weakness of the Popper view is the notion that science progresses linearly. Kuhn introduced the notion of *paradigm shift* to show that scientific progress at some times is non linear and requires a revolution to make progress. In this thesis we want to assess the value of the architecting method for industrial application. The use of a hypothesis and evaluation criteria is less rigid than the Popper approach, but at least it supports an open debate about the merits of the method.

The first step is to make research question and hypothesis explicit. After sufficient research the heuristics and principles will become visible, which can be very powerful means to capture generic know-how, see [68] for an extensive collection of systems architecting heuristics. A nice overview is given by Pidwirny [64],

using characteristics such as *neutral* and *unbiased*.

The next step is to substantiate the benefits of proposed methods with facts and analysis. The last step is to strive for validation. For many soft issues validation will be an unreachable ideal. Increasing the plausibility is then the maximum that can be achieved.

These steps together contribute to the building of a body of know-how (as all sciences do), of which a significant part will be based on case descriptions.

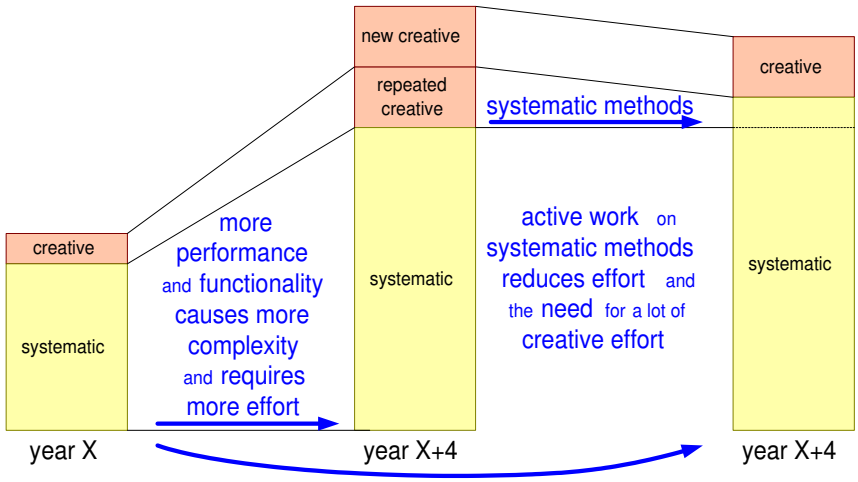


Figure 5.5: A scientific base is required to cope with the growing system effort. The scientific base provides a systematic approach that helps to solve known types of problems with less, more systematic, effort.

The relevance for the product creation companies is that the increasing effort of creating more powerful, but complex systems, is kept manageable. The ratio between the amount of systematic work, engineering, and the amount of creative/chaotic work should preferable stay the same. Due to the increasing complexity, in both hard and soft issues, this ratio will worsen if we are not able to make part of the system work more systematic.

Figure 5.5 shows the amount of systematic work and creative work. In the electronics industry the effort to create new circuits increases exponentially, more or less following Moore's Law. The phenomenon that the product needs and possibilities increase faster than our design know-how is known as the productivity gap, see for example [42]. The first bar shows the amount of systematic work at the bottom and the creative work at the top. The new development shown in the

second bar, taking place several years later, in this example four years, requires about twice the amount of work. If we do not develop the system discipline a lot of the future system work will still be done in "trial and error" mode, represented by the *repeated creative* work. The new functionality, performance and complexity challenges also require *new creative* work. If the creative work of the past can be captured in more systematic approaches then the *repeated creative* work is transformed in less *systematic* work, as shown in the third bar.

One of the symptoms for this trend of increasing creative work is the relative increase of the integration period and integration effort. The lack of a systematic approach in the early design phases is solved by applying a lot of creativity in solving the problems during integration. This effect is visible in complex systems, such as MRI scanners, wafersteppers, and video processing platforms.

The message behind this figure is that product creation will always have a creative component. Providing a scientific base will never remove the need for human creativity. A scientific base will enable the effective use of the creative talent, not wasting it on problems that could have been solved in a systematic way.

Figure 5.5 suggests an incremental increase of creation effort. Many products, such as cardiovascular X-ray systems, wafersteppers, and televisions show such exponential growth of the effort. When developing system architecting methods the ambition should be to develop also the development of system design and implementation methods that *decrease* the desired effort. Once the know-how is captured in methods a next step in support can be made by further automation and supporting tools. Systematizing know how precedes automation and tooling.

5.4 Architecting Research Method

This thesis is based on research by means of the conventional hypothesis (see Section 6.3) and evaluation method, complemented by case descriptions (Part III). The research starts with a research question, described in Section 6.2 that after some exploration work is used to formulate a hypothesis. The hypothesis is next assessed to be valid or invalid by means of criteria, see Section 6.4.

The research method and the architecting methods are very abstract entities. These methods are illustrated by case descriptions. Specific case descriptions make it possible to capture the experience of the otherwise rather generic methods. The case descriptions describe parts of actual system architectures.

In the human sciences case descriptions are one of the major research meth-

ods [70]. Theory in these sciences define many abstract concepts that are difficult to make precise. Case descriptions support the definition of the concepts. At the same time, they complement the abstract concept definitions, by being very specific, thereby helping to clarify and to educate.

5.5 Distance between Industrial Practice and Scientific Research

The main challenge in the research of architecting methods is to bridge the distance between the pragmatic world of product creation in the industrial context and the scientifically sound research of architecting methods. Figure 5.6 shows the distance between the practitioners and the scientific foundation as an abstraction hierarchy.

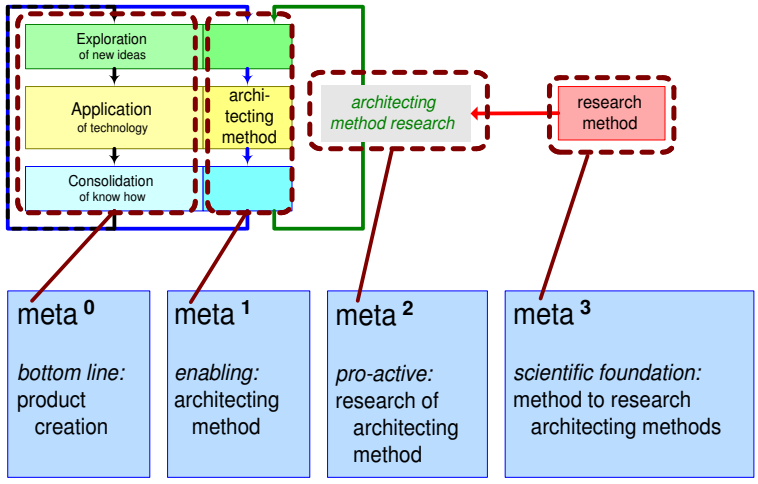


Figure 5.6: Moving in the *meta* direction. Research of architecting methods is two steps of indirection away from the bottom line of product creation. The scientific foundation for this work is another indirection step

The status quo in *systems* architecting is that most architects learn by trial and error[2]. These architects are directly working in the product creation process,

[2] A systematic foundation for *systems* architecting is lacking in the companies I have worked for. Most companies do have extensive process handbooks and quality assurance handbooks, covering documentation, verification, project management, and many more issues. However, the multidisci-

where the bottom line is to create successful products.

The approach taken in *architecting* can be abstracted into an *architecting method*; this is the first step in the *meta*-direction. Doing systematic *research of architecting methods* is a second step in the *meta*-direction. The definition of a *research method* (to investigate *architecting methods*) provides the systematic research with a scientific foundation: the third step in the *meta*-direction. These three levels of abstractions illustrate the different worlds of practitioners and researchers.

The drive behind this thesis is the assumption that building a scientifically founded body of knowledge will improve product creation effectiveness *directly* or *indirectly*. An example of *indirect* improvement by means of rational *design* methods is described by Parnas and Clements in "A Rational Design Process: How and Why to Fake It" [62]. The *rational design process* is in the industrial practice used *indirectly* in the later phases of the product creation for documentation and communication.

5.6 Research Environment

In this thesis architecting methods are studied by a retrospective analysis of a finished industrial product development. This way of working, shown as consolidation in Figure 5.2, makes knowledge that is obtained in the past explicit. This knowledge is consolidated to make it accessible for other people. This way of working does not work for *active* research of architecting methods, where we want to study the effects of potential method improvements.

Figure 5.7 shows multiple environments that can be used to study architecting methods. This thesis is based on *research by analysis* shown at the left hand side. A promising research environment is the *industry as laboratory*. Research of architecting in the limited scope of research laboratories is shown as *trial in research environment*. *Courses* and *workshops* provide an environment to obtain additional feedback on architecting methods.

The *Industry as Laboratory* [65] approach is based on an intimate collaboration of researchers and practitioners. The Embedded Systems Institute uses this

---

plinary specification and design at *system* level is left open.

I have made visits to many other companies, explicitly asking for their systems architecting approach and how they develop systems architects. I did not find any systematic foundation at *system* level in any of these companies. The companies I visited are working in the telecommunication fields, computer industry, and electronics industry.

See Chapter 4 for other work done in this area.

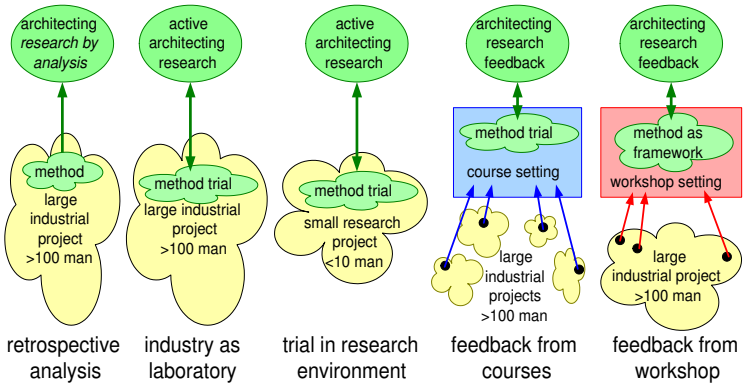


Figure 5.7: Obtaining practical case data of architecting methods from multiple sources

model as the basis for research [69]. The industrial environment is used to try out architecting methods. In the industrial environment the typical time and resource pressures, and the larger size (one or two orders of magnitude larger than in typical research projects) are inherently realistic for the industrial context.

Some architecting methods are also explored in *research projects*. For instance, *story telling*, as described in Chapter 11, is used in ambient intelligence projects. The *CAFCR* model has been used many times in research workshops as a framework. Project reviews and workshop evaluations provide feedback on the architecting method for this *research project* environment.

Partial architecting methods (for instance *story telling* again) are also used in course settings, where they are applied to many different systems, ranging from silicon chips to Cardiovascular X-ray systems. More than 300 designers and architects have participated in Systems Architecting courses, using partial methods for tens of different systems. This provides valuable feedback of these methods when applied to real systems. See [51] for the course program. The entire course material, including exercises, can be found at: `http://www.extra.research.philips.com/natlab/sysarch/SARCH.html`.

The CAFCR method is also used within Philips as a framework for performing architecture workshops. External and internal project stakeholders present during the workshops use (parts of) the CAFCR method as a means to structure their workshop. The evaluation at the end of a workshop provides feedback for the architecting method. In Chapter 19 the evaluations from many workshops are discussed.

Chapter 6

Research Question and Hypothesis

6.1 Introduction

The starting point for the investigation of architecting methods in this thesis is the research question, articulated in Section 6.2. An hypothesis is formulated in Section 6.3. The criteria to validate the hypothesis are defined in Section 6.4. Section 6.5 summarizes all three aspects in a single overview, and shows how the different parts of the thesis fit together in this thesis.

6.2 Research Question

Figure 6.1 shows the annotated research question. The core research question is *what architecting methods enable the creation of successful products*. This core question is more focused by adding a more specific environment: *a dynamic market* and a *heterogeneous industrial context*. The product category is also narrowed down by looking only at *technology and software intensive* products.

Successful products are products that satisfy the customers and result in a thriving business. In present-day economy this means that time plays a dominant role, products must be *in time*. The economic reality and the hefty competition force economic constraints on the created product and its timing. This economic reality requires pragmatism in the architecting methods. Many academic methods, however, suffer from a mismatch with these economic and time constraints. For instance formal verification systems work well for small well-defined prob-

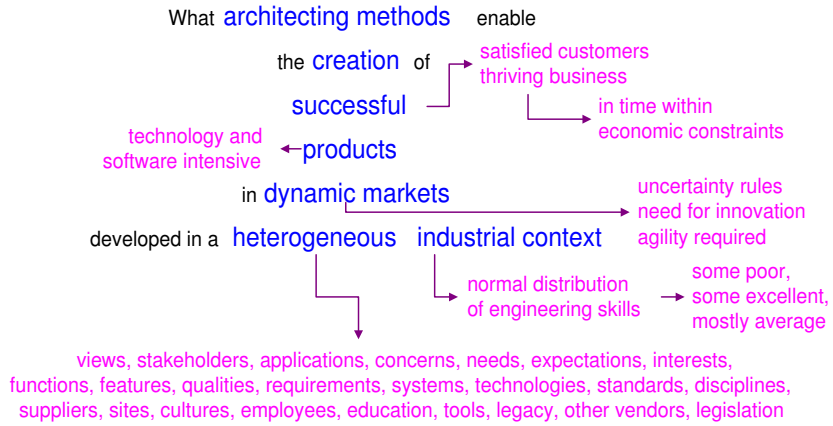


Figure 6.1: Research question

lems, but require too much time and skills to be useful in larger, more uncertain, problems.

Most product areas become more and more dynamic: customers, competitors, and other stakeholders interact in complex ways, with a lot of uncertainty. Active commercial product lifetimes have decreased from years to months. Globalization enables unexpected competitors to enter the market, based on low-cost labor and huge work-forces. Constant innovation is required to stay competitive. To follow the rapid market changes agile procedures and organizations are required.

The context in which we operate is characterized by an increasing variability and complexity. Products should fit in a very heterogeneous world. The heterogeneity is present in aspects such as: *views, stakeholders, applications, concerns, needs, expectations, interests, functions, features, qualities, requirements, systems, technologies, standards, disciplines, suppliers, sites, cultures, employees, education, tools, legacy, other vendors, legislation.*

The industrial context in which the methods have to be used has a population of engineers with a normal distribution of engineering skills and intellect. Some have poor skills, some have excellent skills, but most engineers have average skills. This is a severe constraint on the architecting methods. Some very nice methods are too difficult to apply in practical organizations. Note that the research question is what methods enable the product creation in the *industrial context*. This does not imply that the constraint is that they should fit entirely in the existing crew. Crew and method should be matched, but the degrees of

freedom in composing a PCP team in an industrial context are quite limited.

6.3 Hypothesis

The variability of products being created is so large, that **one** all encompassing method is impossible. The dynamic range in requirements spans many orders of magnitude. For example requirements for power consumption, weight, and processing needs differ a factor of 1000 for products such as televisions and GSM cellphones. It is more feasible to grow a rich collection of submethods than to develop a single all encompassing method. Submethods are methods that address a smaller part of the problem. This step moves the problem to another area: how to combine multiple submethods in a useful way?

A **rich collection** of **submethods** fitting in a **multi-view framework**

complemented with **reasoning methods** enables **successful** architecting of

technology and software intensive complex systems in heterogeneous environments

by means of **generic insights** grounded in **specific facts**

Figure 6.2: Hypothesis

The hypothesis, as shown in Figure 6.2 formulates the need for a *multi-view framework*. The submethods must fit in the multi-view framework. *Reasoning methods* are needed to cope with multiple submethods.

The claim is that this combination of a *rich collection of submethods*, *multi-view framework*, and *reasoning methods* enables *the successful architecting of technology and software intensive complex systems in heterogeneous environments*.

The generic term *product* in the research question is replaced by the more focused notion of *technology and software intensive systems*. The research described in this thesis has been limited to embedded systems. Embedded systems are systems with embedded computing hardware and software that have interaction with the physical world. This interaction with the physical world is technology intensive, for example actuator technology and sensor technology.

In addition two crucial characteristics of architecting work are added to the claim: the use of *generic insights* grounded in *specific facts*. These two character-

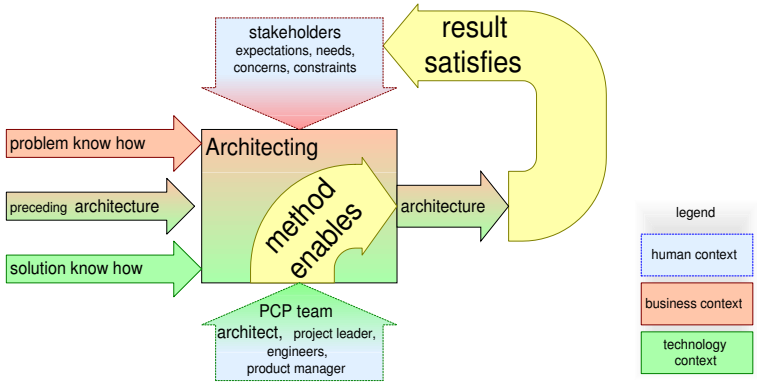


Figure 6.3: The hypothesis is valid if *successful* architecting is *enabled*.

istics seem to be contradictory: generic insights are often interpreted as ignoring the details (=specific facts); some details, however, are often needed to appreciate the essence captured in the generic insight.

Architecting methods need sufficient genericity to have impact. Architects will lose overview when they have to specify every product detail. The challenge is to extract the essence from specific facts in such a way that powerful and trustworthy generic insights are created.

Many product developments fail in combining the specific facts and the generic insights. Discussions during the SARCH courses [19] often show organizations inside and outside Philips where the PCP teams spend all their time in details. The policy makers in these same organizations are disconnected from the rest of the PCP team. The PCP team is working on *specific* details, while the policy makers are working on *generic* insights, but the two worlds are disconnected. The consequence of the disconnection is that product innovations fail. Small improvements are made by the PCP team, but the larger changes fail because important details have not been taken into account.

Figure 6.3 addresses the term *successful* in the hypothesis. This is done in a two-step approach from PCP team to stakeholders. The PCP team is successfully enabled by an architecting method if the use of the method resulted in the creation of a successful architecture. An architecture is successful if the stakeholders are satisfied with the result.

6.4 Criteria

Figure 6.4 shows the criteria to be used, based on the two step approach shown in Figure 6.3.

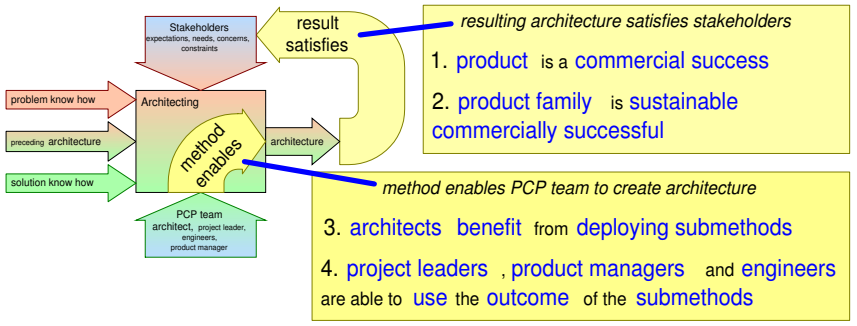


Figure 6.4: From hypothesis to criteria

The resulting architecture satisfies the stakeholders is indirectly verified by measuring the *1. short term commercial success of the product* and the *2. sustainability of this commercial success in the following product family*. The underlying assumption is that satisfied customers buy more products and motivate other customers to buy this satisfactory product. Dissatisfied customers have a negative impact on the sales.

In order to exclude incidental success, the long term commercial success is also required. This long term success can only be measured by means of follow-on products. The active commercial lifetime of products (1 to 2 years) is too short to measure long term commercial success with the product itself. The follow-on product family, based on the same architecture and architecting method, is used instead.

The architecting method enables the PCP team to create a successful architecture is sharpened by defining two criteria. The first criterion is for the architect as primary user: the architect(s) must be able to use the submethods to achieve a good architecture. Note that the hypothesis mentions *submethods complemented with reasoning methods*. The criterion for success is not that all submethods are useable, but the criterion is that *3. the architect benefits from the collection of submethods*.

The second criterion to assess is the enabling of the product creation team, especially the (non-architect) members of the PCP team: *4. the outcome of the architecting method must be usable for the other members of the Product Creation*

Team (or PCP Team): project leaders, product managers and engineers.

The quality of the design of the product contributes to the sustainability of the product. The quality of design is also one of the measures for the support that the method provides to architects. Another measure for the support is the integration: How well was integration supported by the chosen integration?

6.5 Summary

In this chapter we have discussed the research question, the objectives of the architecting method, the hypothesis about the architecting method improvements, and the criteria to evaluate the method. Figure 6.5 shows the summary of this chapter: research question, hypothesis and criteria.

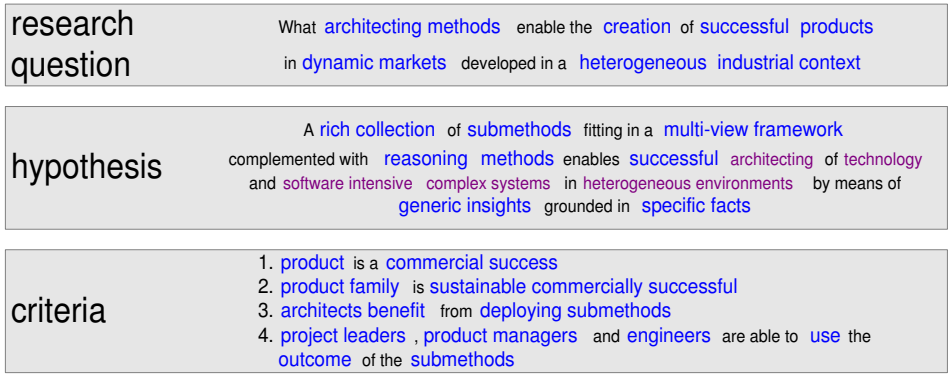


Figure 6.5: Overview of research question, hypothesis and criteria.

Part II will show the theory of the architecting method. Part III describes the case that is used for the evaluation. In Part IV, in chapters 18 and 19, the hypothesis and criteria are used for the evaluation.

Part II

Theory of CAFCR and Threads of Reasoning

Chapters in Part II:

Chapter 7

Basic Methods

7.1 Introduction

The basic methods used by system architects are covered by a limited set of very generic patterns:

- Viewpoint hopping, looking at the problem and (potential) solutions from many points of view, see Section 7.2.
- Decomposition, breaking up a large problem into smaller problems, introducing interfaces and the need for integration, see Section 7.3.
- Quantification, building up understanding by quantification, from order of magnitude numbers to specifications with acceptable confidence levels, see Section 7.4.
- Decision making when lots of data is missing, see Section 7.5.
- Modelling, as means of communication, documentation, analysis, simulation, decision making and verification, see Section 7.6.
- Asking Why, What, How, Who, When, Where questions, see Section 7.7.
- Problem solving approach, see Section 7.8.

Besides these methods the architect needs lots of “soft” skills, to be effective with the large amount of different people involved in creating the system, see [68].

7.2 Viewpoint Hopping

The architect is looking towards problems and (potential) solutions from many different viewpoints. A small subset of viewpoints is visualized in Figure 7.1, where the viewpoints are shown as stakeholders with their concerns.

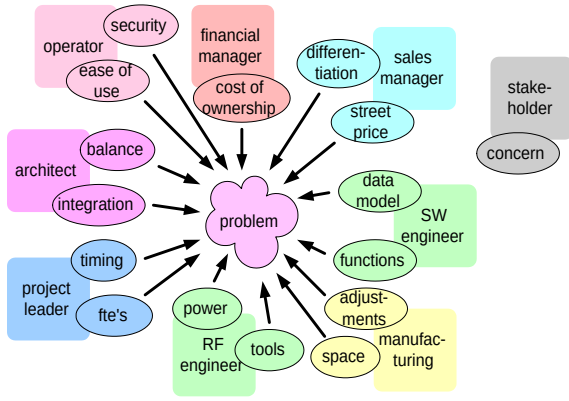


Figure 7.1: Small subset of stakeholders, concerns and viewpoints

The architect is interested in an overall view on the problem, where all these viewpoints are present simultaneously. The limitations of the human brains force the architect to create an overall view by quickly alternating the individual viewpoints. The order in which the viewpoints are alternated is chaotic: problems or opportunities in one viewpoint trigger the switch to a related viewpoint. Figure 7.2 shows a very short example of viewpoint hopping. This example sequence can take anywhere from minutes to weeks. In a complete product creation project the architect makes thousands[1] of these viewpoint changes.

The system description and implementation span a significant dynamic range. At the highest abstraction level a system can be characterized by its core function and the key performance figure. Via multiple decomposition steps the description is detailed to units that can be engineered. The implementation shows orders of magnitude more details. The source description of today's products is in the order of millions lines of code[2]. The source description expands via synthesis into an

[1] Based on observations of other architects and own experience.

[2] In 2003 the software figures for MR scanners, wafersteppers and televisions are all between 1 and 10 million lines of source code. Source code in the broad sense: all formalized definitions, that are created by humans and maintained and tested. Generated code is not counted.

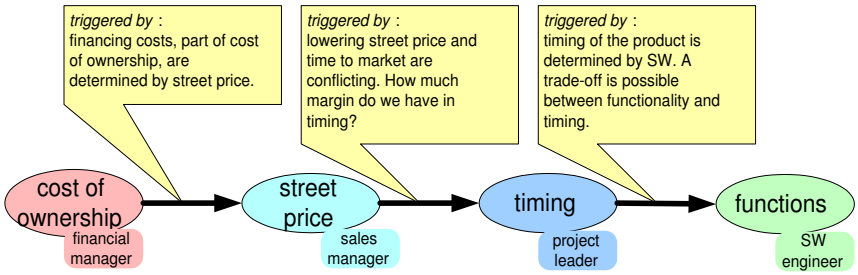


Figure 7.2: Short example of viewpoint hopping

order magnitude more wires, gates, transistors and bytes. The amount of states in actual operation is again orders of magnitude larger.

Figure 7.3 shows this dynamic range. At the left hand abstraction levels in the creation life-cycle are shown. Both for hardware and software the typical entities at the different layers of abstraction are shown.

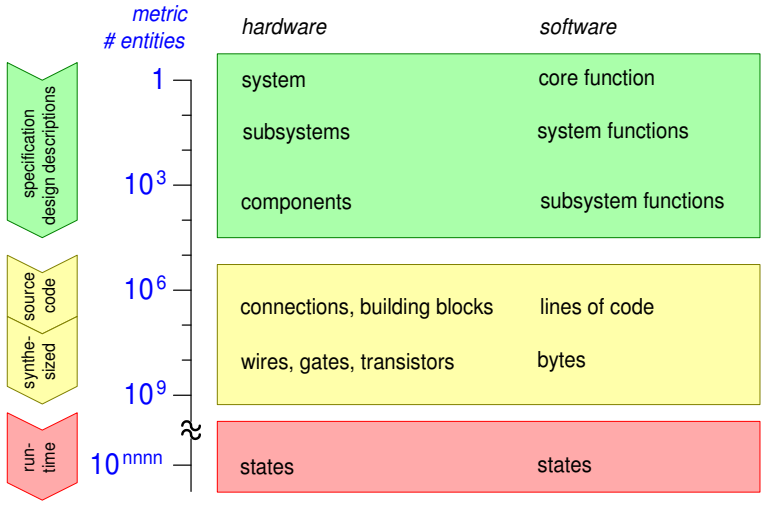


Figure 7.3: Dynamic range of description and implementation in a product

The viewpoints and dynamic range of abstraction create a huge space for exploration. Systematic scanning of this space is way too slow. An architect is using two techniques to scan this space, that are quite difficult to combine: open perceptive scanning and scanning while structuring and judging. The open perceptive

mode is needed to build understanding and insight. Early structuring and judging is dangerous because it might become a self-fulfilling prophecy. The structuring and judging is required to reach a result in a limited amount of time and effort. See Figure 7.4 for these 2 modes of scanning.

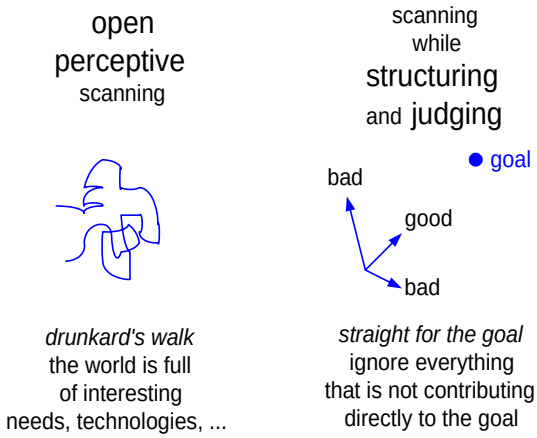


Figure 7.4: Two modes of scanning by an architect

The scanning approach taken by the architect can be compared with *simulated annealing methods* for optimization. An interesting quote from the book "Numerical Recipes in C; The Art of Scientific Computing"[66], about comparing optimization methods is:

> Although the analogy is not perfect, there is a sense in which all of the minimization algorithms thus far in this chapter correspond to rapid cooling or quenching. In all cases, we have gone greedily for the quick, nearby solution: From the starting point, go immediately downhill as far as you can go. This, as often remarked above, leads to a local, but not necessarily a global, minimum. Nature's own minimization algorithm is based on a quite different procedure...

The exploration space is not exclusively covered by the architect(s), engineers will cover a large part of this space. The architect will focus mostly on the higher abstraction layers, *but* sufficient sampling of the lower layers is important to keep the higher layers meaningful.

7.3 Decomposition and Integration

The architect applies a reduction strategy by means of decomposition over and over. Decomposition is a very generic principle, that can be applied for many different problem and solution dimensions. Martin [46] uses the phrase development layers when the decomposition principle is applied on a system.

Whenever something is decomposed the resulting components will be decoupled by interfaces. The architect will invest time in interfaces, since these provide a convenient method to determine system structure and behavior, while separating the inside of these components from their external behavior.

The true challenge for the architect is to design decompositions, that in the end will support an integration of components into a system. Most effort of the architect is concerned with the integrating concepts, how do multiple components work together?

Many stakeholders perceive the decomposition and the interface management as the most important contribution. In practice it is observed that the synthesis or integration part is more difficult and time consuming.

7.4 Quantification

The architect is continuously trying to improve his understanding of problem and solution. This understanding is based on many different interacting insights, such as functionality, behavior, relationships et cetera. An important factor in understanding is the **quantification**. Quantification helps to get grip on the many vague aspects of problem and solution. Many aspects can be quantified, much more than most designers are willing to quantify. Thomas Gilb stresses the importance of quantification and estimation, see for instance [26].

The precision of the quantification increases during the project. Figure 7.5 shows the stepwise refinement of the quantification. In first instance it is important to get a feeling for the problem by quantifying orders of magnitude. For example:

- How large is the targeted customer population?
- What is the amount of money they are willing and able to spend?
- How many pictures/movies do they want to store?
- How much storage and bandwidth is needed?

The order of magnitude numbers can be refined by making back of the envelop calculations, making simple models and making assumptions and estimates. From

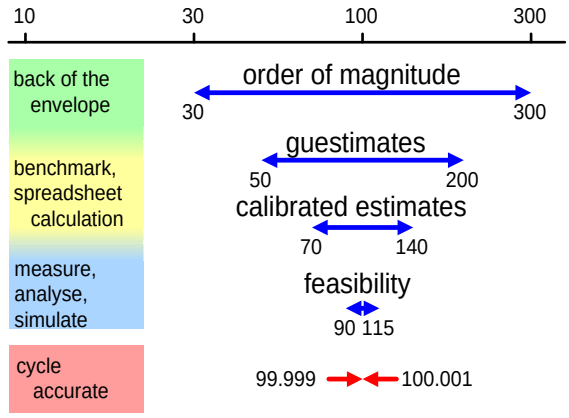


Figure 7.5: Successive quantification refined

this work it becomes clear where the major uncertainties are and what measurements or other data acquisitions will help to refine the numbers further.

At the bottom of Figure 7.5 the other extreme of the spectrum of quantification is shown. In this example cycle-accurate simulation of video frame processing results in very accurate numbers. It is a challenge for an architect to bridge these worlds.

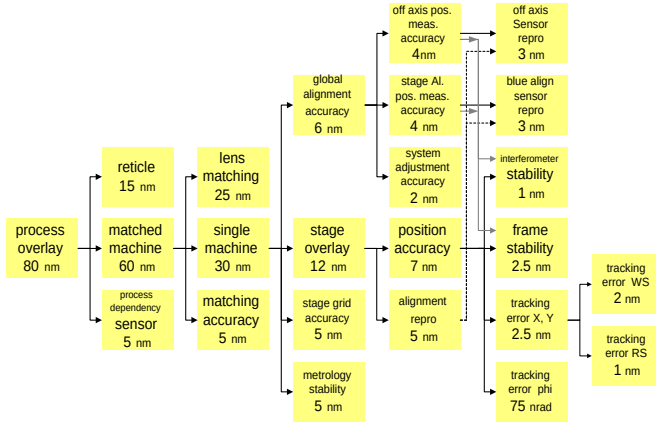


Figure 7.6: Example of a quantified understanding of overlay in a waferstepper

Figure 7.6 shows a graphical example of an “overlay” budget for a waferstepper. This figure is taken from the *System Design Specification* of the ASML

TwinScan system, although for confidentiality reasons some minor modifications have been applied. This budget is based on a model of the overlay functionality in the waferstepper. The budget is used to provide requirements for subsystems and components. The actual contributions to the overlay are measured during the design and integration process, on functional models or prototypes. These measurements provide early feedback of the overlay design. If needed the budget or the design is changed on the basis of this feedback.

7.5 Coping with Uncertainty

The architect has to make decisions all the time, while most substantiating data is still missing. On top of that some of the available data will be false, inconsistent or interpreted wrong.

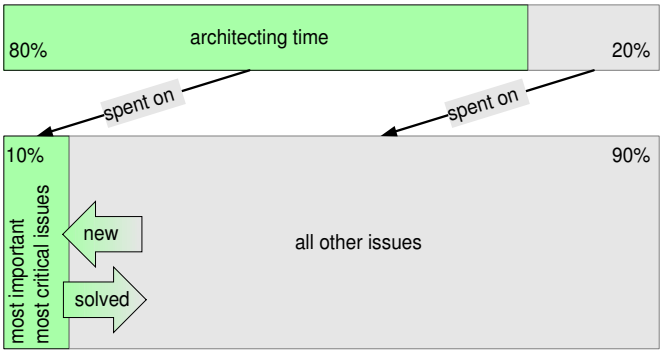


Figure 7.7: The architect focuses on important and critical issues, while monitoring the other issues

An important means in making decisions is building up insight, understanding and overview, by means of structuring the problems. The understanding is used to determine important (for the product use) and critical (with respect to technical design and implementation) issues. The architect will pay most attention to these *important* and *critical* issues. The other issues are monitored, because sometimes minor details turn out to be important or critical issues. Figure 7.7 visualizes the time distribution of the architect: 80% of the time is spent on 10% of the issues. The well known 80/20 rule matches well with my own observations of how system architects spend their time. The number of important issues that are addressed in the 80% of the time is also based on personal observations.

7.6 Modeling

modeling is one of the most fundamental tools of an architect. Gilb [26] defines a model as

> A model is an artificial representation of an idea or a product. The representation can be in any useful format including specifications, drawings and physical representations. (Gilb glossary of concepts)

Lieberman [44] explains the difficulty of making good models for human use.

In summary we can say that models are used to obtain insight and understanding, and that models serve a clear purpose. At the same time the architect is always aware of the (over)simplification applied in every model. A model is very valuable, but every model has its limitations, imposed by the simplifications.

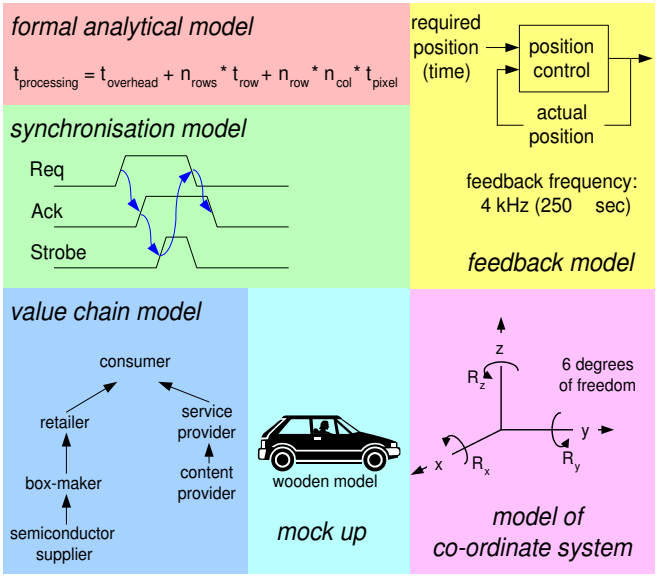


Figure 7.8: Some examples of models

Models exist in a very rich variety. A few examples of models are shown in Figure 7.8.

Models have many different manifestations. Figure 7.9 shows some of the different types of models, expressed in a number of adjectives.

Models can be *mathematical* (a mature field, see for instance [14]) expressed in formulas, they can be *linguistic*, expressed in words, or they can be *visual*, cap-

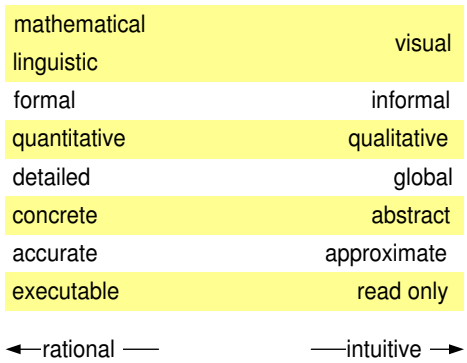


Figure 7.9: Types of models

tured in diagrams. A model can be formal, where notations, operations and terms are precisely defined, or it can be informal, using plain English and sketches. Quantitative models use meaningful numbers, allowing verification and judgements. Qualitative models show relations and behavior, providing understanding. Concrete models use tangible objects and parameters, while abstract models express mental concepts. Some models can be executed (as a simulation), while other models only make sense for humans reading the model.

7.7 WWHWWW

All "W" questions are an important tool for the architect. Figure 7.10 shows the useful starting words for questions to be asked by an architect.

Why	Who
What	When
How	Where

Figure 7.10: The starting words for questions by the architect

Why, **what** and **how** are used over and over in architecting. Why, what and how are used to determine objectives, rationale and design. This works highly recursively, a design has objectives and a rationale and results in smaller designs

that again have objectives and rationales. Figure 7.11 shows that the recursion with **why** questions broadens the scope, and recursion with **how** questions opens more details in a smaller scope.

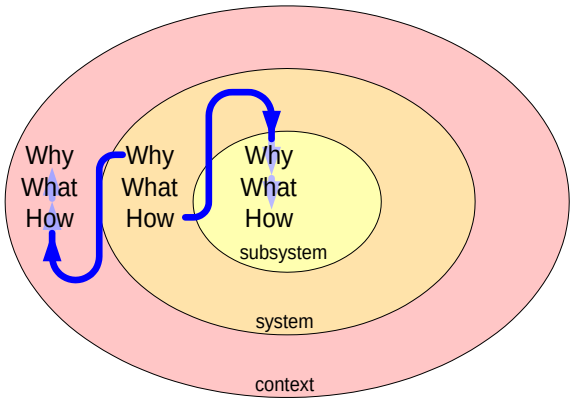


Figure 7.11: Why broadens scope, How opens details

Who, **where** and **when** are used somewhat less frequently. Who, where and when can be used to build up understanding of the context, and are used in cooperation with the project leader to prepare the project plan.

7.8 Decision Making Approach in Specification and Design

Many specification and design decisions have to be taken during the product creation process. For example, functionality and performance requirements need to be defined, and the way to realize them has to be chosen. Many of these decisions are interrelated and have to be taken at a time when many uncertainties still exist, see section 7.5. The need for problem solving and decision making techniques is clearly recognized in companies like Philips and ASML. Quality improvement programs in these companies include education in these techniques. The approach described in this section is based on the techniques promoted by the quality improvement programs.

An approach to make these decisions is the flow depicted in Figure 7.12. The decision process is modeled in four steps. An understanding of the problem is created by the first step *problem understanding*, by exploration of problem and

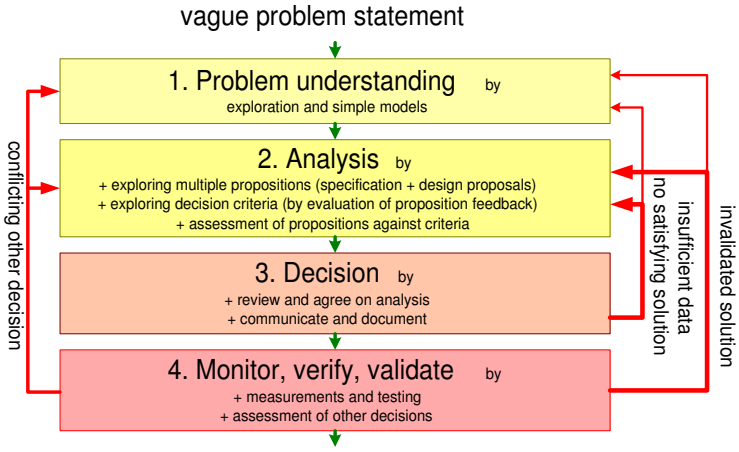


Figure 7.12: Flow from problem to solution

solution space. Simple models, in problem space as well as in solution space, help to create this understanding. The next step is to perform a somewhat more systematic *analysis*. The analysis is often based on *exploring multiple propositions*. The third step is the *decision* itself. The analysis results are reviewed, and the decision is documented and communicated. The last step is to *monitor, verify and validate* the decision.

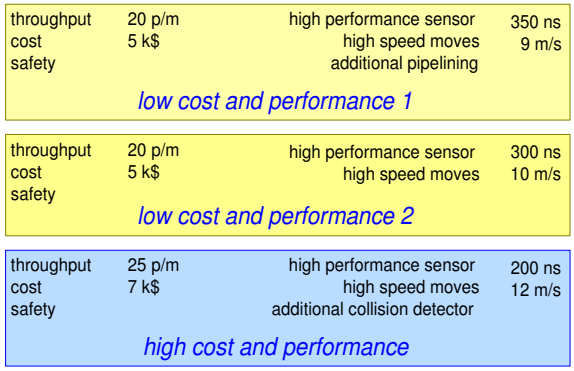


Figure 7.13: Multiple propositions

The *analysis* involves multiple substeps: *exploring multiple propositions*, *exploring decision criteria* and *assessing the propositions against the criteria*. A

proposition describes both specification (**what**) and design (*how*). Figure 7.13 shows an example of multiple propositions. In this example a high performance, but high cost alternative, is put besides two lower performing alternatives. Most criteria get articulated in the discussions about the propositions: “I think that we should choose proposition 2, because...”. The *because* can be reconstructed into a criterion.

The decision to chose a proposition is taken on the basis of the analysis results. A review of the analysis results ensures that these results are agreed upon. The decision itself is documented and communicated[3]. In case of insufficient data or in absence of a satisfying solution we have to back track to the *analysis* step. Sometimes it is better to revisit the problem statement by going back to the *understanding* step.

Taking a decision requires a lot of follow up. The decision is in practice based on partial and uncertain data, and is based on many assumptions. A significant amount of work is to monitor the consequences and the implementation of the decision. Monitoring is partially a *soft skill*, such as actively listening to engineers, and partially a *engineering activity*, such as measuring and testing. The consequence of a measurement can be that the problem has to be revisited, because the solution is invalidated. An invalidated solution returns the process to the *understanding* step in case of serious mismatches (’apparently we don’t understand the problem at all’). In case of smaller mismatches the process returns to the *analysis* step.

The implementation of taken decisions can be disturbed by later decisions. This problem is partially tackled by requirements traceability, where known interdependencies are managed explicitly. In the complex real world the amount of dependencies is almost infinite, that means that the explicit dependability specifications are inherently incomplete and only partially understood. To cope with the inherent uncertainty about dependabilities, an open mind is needed when screening later decisions. A conflict caused by a later decision triggers a revisit of the original problem.

The same flow of activities is used recursively at different levels of detail, as shown in Figure 7.14. A *system* problem will result in a system design, where many design aspects need the same flow of problem solving activities for the subsystems. This process is repeated for smaller scopes until termination at problems that can be solved directly by an implementation team. The smallest scope of termination is denoted as *atomic* level in the figure. Note that the more detailed

[3]This sounds absolutely trivial, but unfortunately this step is performed quite poorly in practice.

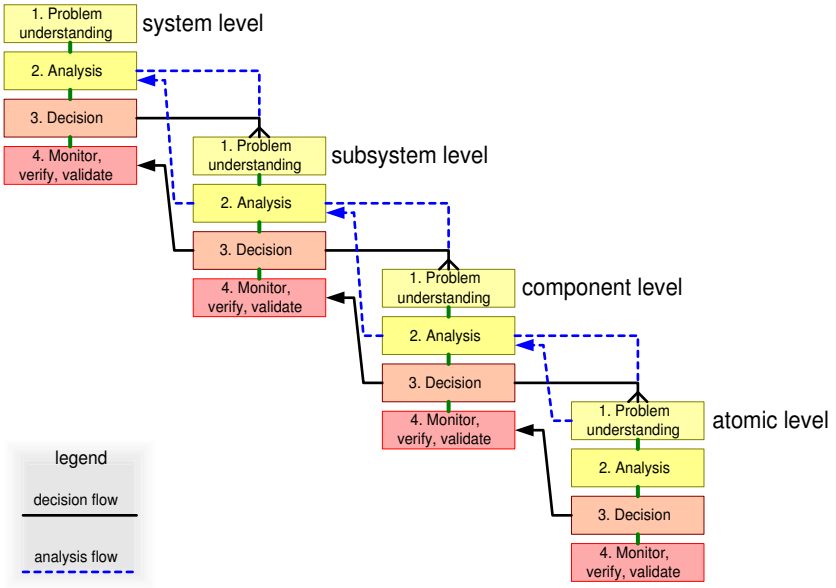


Figure 7.14: Recursive and concurrent application of flow

problem solving might have impact on the more global decisions.

Chapter 8

Submethods in the CAF Views

8.1 Introduction

This chapter describes the submethods in the *Customer objectives*, *Application* and *Functional* views.

Section 8.2 describes a submethod to identify key drivers and to relate the key drivers to product requirements.

Section 8.3 mentions submethods that help in positioning the business of the customer in the context: *analysis of the value chain*, *analysis of business models*, *creation of a map of competitors and complementers*, *application context diagram* and *identification and articulation of the stakeholders and concerns*.

The basic methods modeling and quantification from Chapter 7 are used in Section 8.4 to identify useful models in the customer world.

Section 8.5 describes *use cases* as description and communication means for behavioral as well as quantitative characteristics.

The leading document in the product creation process is the *system specification*, which is discussed in Section 8.6.

Section 8.7 provides an overview of all the submethods discussed in this chapter, and positions the submethods in the CAFCR views.

8.2 Key Drivers

The essence of the objectives of the customers can be captured in terms of customer key drivers. The key drivers provide direction to capture requirements and to focus the development. This method has been successfully deployed to focus

the product creation process of ASML [7]. A closely related method is "Goal-oriented design" [34], which is also based on analysis of the customer needs and goals in a hierarchical fashion.

The key drivers in the customer objectives view will be linked with requirements and design choices in the other views. The key driver submethod gains its value from relating a few sharp articulated key drivers to a much longer list of requirements. By capturing these relations a much better understanding of customer and product requirements is achieved. In Quality Function Deployment [67], where the term *benefits* is used for key driver, the link between *benefits*, *engineering requirements* and *design concepts* is emphasized.

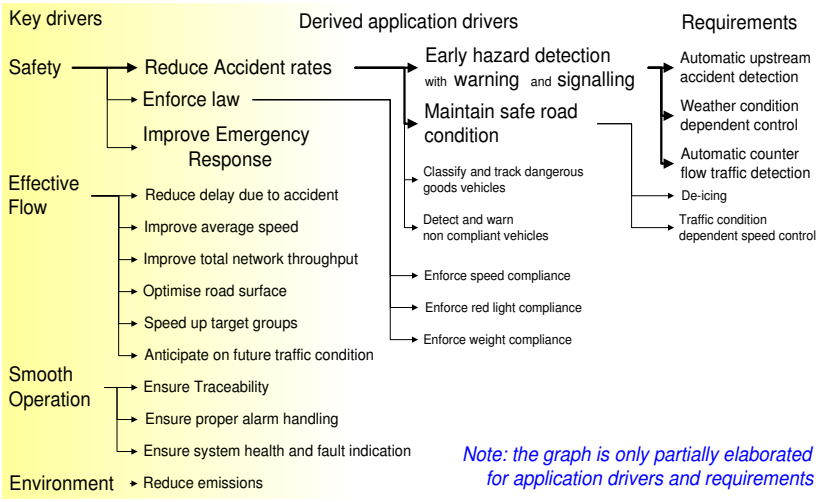


Figure 8.1: Example of the four key drivers in a motorway management system

Figure 8.1 shows an example of key drivers for a motorway management system, an analysis performed at Philips Projects in 1999.

Figure 8.2 shows a submethod how to obtain a graph linking key drivers to requirements. The first step is to define the scope of the key driver graph. For Figure 8.1 the customer is the motorway management operator. The next step is to acquire facts, for example by extracting functionality and performance figures out of the product specification. Analysis of these facts recovers implicit facts. The requirements of an existing system can be analyzed by repeating *why* questions. For example: "Why does the system need *automatic upstream accident detection*?". The third step is to bring more structure in the facts, by building a

• Define the scope specific.	in terms of stakeholder or market segments
• Acquire and analyze facts	extract facts from the product specification and ask why questions about the specification of existing products .
• Build a graph of relations between drivers and requirements by means of brainstorms and discussions	where requirements may have multiple drivers
• Obtain feedback	discuss with customers , observe their reactions
• Iterate many times	increased understanding often triggers the move of issues from driver to requirement or vice versa and rephrasing

Figure 8.2: Submethod to link key drivers to requirements, existing of the iteration over four steps

graph, which connects requirements to key drivers. A workshop with brainstorms and discussions is an effective way to obtain the graph. The last step is to obtain feedback from customers. The total graph can have many n:m relations, i.e. requirements that serve many drivers and drivers that are supported by many requirements. The graph is good if the customers are enthusiastic about the key drivers and the derived application drivers. If a lot of explaining is required then the understanding of the customer is far from complete. Frequent iterations over these steps improves the quality of the understanding of the customer's viewpoint. Every iteration causes moves of elements in the graph in driver or requirement direction and also causes rephrasing of elements in the graph.

• Limit the number of key drivers	minimal 3, maximal 6
• Don't leave out the obvious key drivers	for instance the well-known main function of the product
• Use short names, recognized by the customer.	
• Use market/customer specific names, no generic names	for instance replace " ease of use " by "minimal number of actions for experienced users ", or "efficiency " by "integral cost per patient "
• Don' t worry about the exact boundary between Customer Objective and Application	create clear goal means relations

Figure 8.3: Recommendations for applying the key driver submethod

Figure 8.3 shows an additional set of recommendations for applying the key driver submethod. The most important goals of the customer are obtained by limiting the number of key drivers. In this way the participants in the discussion are

forced to make choices. The focus in product innovation is often on differentiating features, or unique selling points. As a consequence, the core functionality from the customer's point of view may get insufficient attention. An example of this are cell phones that are overloaded with features, but that have a poor user interface to make connections. The core functionality must be dominantly present in the graph. The naming used in the graph must fit in the customer world and be as specific as possible. Very generic names tend to be true, but they do not help to really understand the customer's viewpoint. The boundary between the Customer Objectives view and the Application view is not very sharp. When creating the graph that relates *key drivers* to *requirements* one frequently experiences that a key driver is phrased in terms of a (partial) solution. If this happens either the key driver has to be rephrased or the solution should be moved to the requirement (or even realization) side of the graph. A repetition of this kind of iterations increases the insight in the needs of the customer in relation to the characteristics of the product. The **why**, **what** and **how** questions can help to rephrase drivers and requirements. The graph is good if the relations between goals and means are clear for all stakeholders.

8.3 Customer Business Positioning

The position of the customer in the *value chain* and *the business models* [63] deployed by the players in the value chain are important factors in understanding the goals of this customer. The analysis of the *value chain* of the customer and the analysis of the *business models* involved have been deployed as successful submethods in the *Medical Systems*, *Consumer Electronics* and *Semiconductors* product divisions in Philips.

Related to the value chain is the way the customers view their suppliers. The customer sees your company as one of the (potential) suppliers. From the customer's point of view products from many suppliers have to be integrated to create the total solution for his needs. In terms of your own company this means that you have to make a *map of competitors and complementers*, that together will supply the solution to the customer. Figure 8.4 shows an example of a *map of competitors and complementers* in the medical domain taken from [58].

An *application context diagram* is a diagram that shows all systems that can be operational in the application context. The *application context diagram* emphasizes the provided functionality. Many systems in the customer domain have no direct interface with the product under consideration. The interaction of sys-

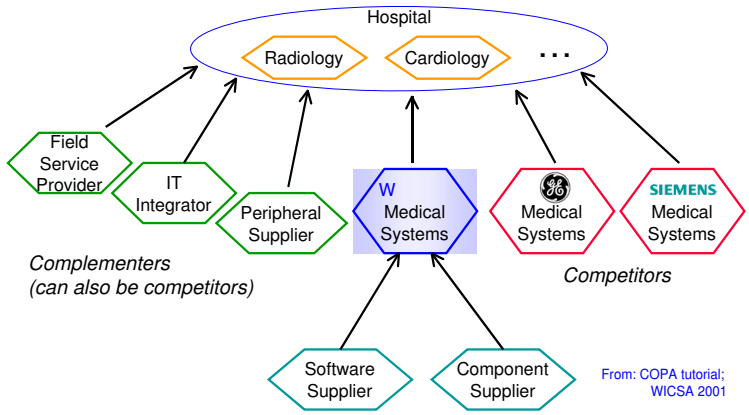


Figure 8.4: Example map of competitors and complementers from the medical domain

tems in the context with the product happens in many cases via human operators. The value of such a diagram is to understand function allocation in the customer context and to understand the value of potential improvements, such as further integration or automation.

Figure 8.5 shows a simple context diagram of the motorway management system of Figure 8.1. Tunnels and toll stations often have their own local management systems, although they are part of the same motorway. The motorway is connecting destinations, such as urban areas. Urban areas have many traffic systems, such as traffic management (traffic lights) and parking systems. For every system in the context questions can be asked, such as:

- is there a need to interface directly (e.g. show parking information to people still on the highway)?
- is duplication of functionality required (measuring traffic density and sending it to a traffic control center) or not?

The *map of competitors and complementers* focuses on economic parties and their roles, while the *application context diagram* focuses on the functional operation in the customer context.

The IEEE 1471 [6] standard about architectural descriptions uses stakeholders and concerns as the starting point for an architectural description. *Identification and articulation of the stakeholders and concerns* is a first step in understanding the application domain. This approach matches very well with the CAFCR

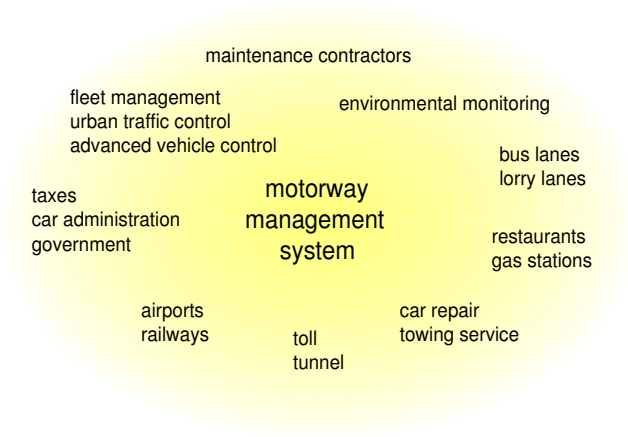


Figure 8.5: Systems in the context of a motorway management system

approach

In practice the *informal* relationships get insufficient attention. In many cases the formal relationships, such as organization charts and process descriptions, are solely used for the analysis of the stakeholders and their concerns. The understanding of the customer context is then incomplete, often causing the failure of the solution. Many organizations function thanks to the unwritten information flows of the social system. Insight in the informal side is required to prevent a solution that does only work in theory.

8.4 Modeling in the Customer World

The customer context can be statically modelled by deploying modeling techniques from the information technology world, for instance entity relationship diagrams [29].

Dynamic models are used to model the logical behavior or the behavior in time. Examples of dynamic models are shown in Figure 8.6:

- flow models that model the flow of goods, people or information
- state diagrams make the dynamics explicit by means of states and state transitions -
- time line that show the events as a function of time

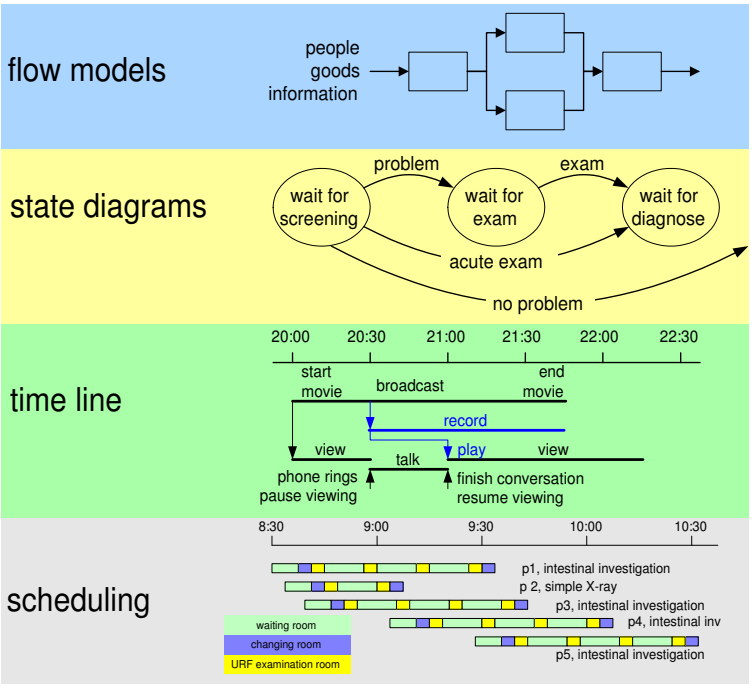


Figure 8.6: Examples of dynamic models

- resource management by means of scheduling models

Seitz [47] discusses the use of *time domain* and *sequence domain* models in electronic circuit design context.

A comparable class of modelling techniques focuses on economical aspects: for instance *Productivity models* and *Cost of ownership models*. The throughput model shown in Figure 8.7 has been used for wafersteppers. This throughput model is internally based on a dynamic model. The throughput model can be used as black box, with a simple parameterized model. In this example the parameters are the *job* that the customer wants to repeat, the *configuration* of the system at the customer side and the *working conditions*. An example of a *Cost of Ownership* (CoO) model is shown in figure 8.8. Combination of productivity and CoO models are used for cost benefit analysis. Gartner [4] has done a lot of work in the area of cost benefit analysis. The Gartner models are well appreciated in industry. These models can be used as started point for modeling the customer world.

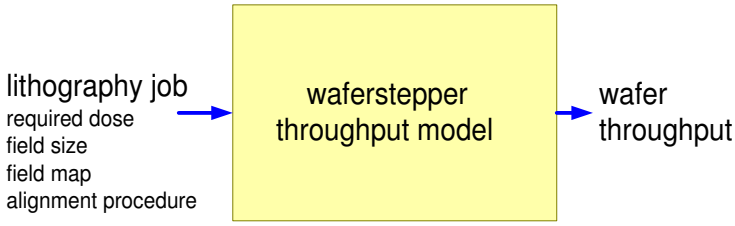


Figure 8.7: A throughput model that estimates the throughput as function of user controlled values

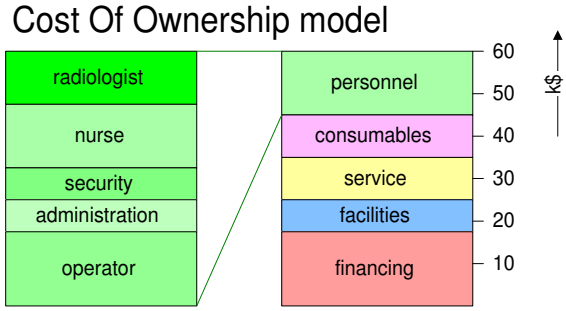


Figure 8.8: Example of a Cost of Ownership model

8.5 Use Cases

Use cases [16] are widely used in software development to describe how the system is used. Most software people use the *use case* submethod only for behavioral descriptions. In embedded systems design this submethod is also very useful for quantitative descriptions of the system, for instance for performance.

Figure 8.9 shows a classification of use cases, with several examples from the Personal Video Recorder (PVR) domain per category. The most typical use of a PVR is to watch movies: find the desired movie and play it. Additional features are the possibility to pause or to stop and to skip forward or backward. The use case description itself should describe exactly the required functionality. The required non-functional aspects, such as performance, reliability and exceptional behavior must be described as well.

Typical use cases describe the core requirements of the products. The boundaries of the product must be described as well. These boundaries can be simply specified (for example, maximum amount of video stored is 20 hours standard

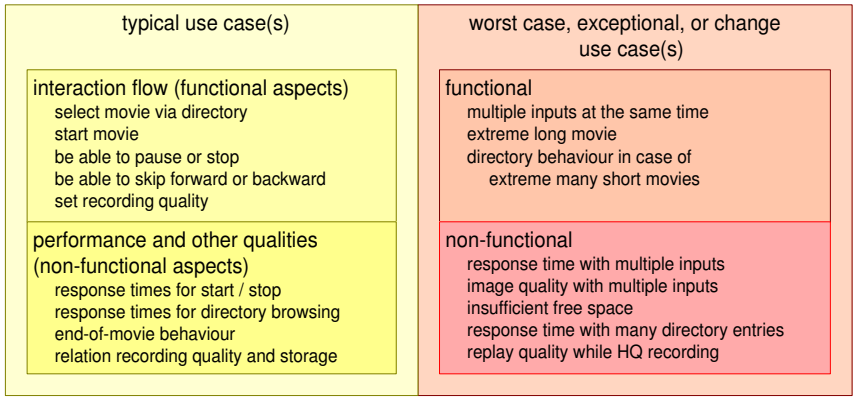


Figure 8.9: Classification of use cases, with several examples from the Personal Video Recorder domain per category.

quality or 10 hours high definition quality) or a set of *worst case* use cases can be used. *Worst case* use cases are especially useful when the boundaries are rather situational dependent. These situations can then be described in the use case.

Exceptional use cases are comparable to worst-case use cases. Exceptions can be described directly (for example, if insufficient storage space is available the recording stops and a message is displayed). Here *exception* use cases are helpful if the exception and the desired exceptional behavior depend on the circumstances.

Change use cases are cases that are deployed in a later phase of the product creation to assess the extendibility and flexibility of the architecture. Change cases address expected functional extensions or performance improvements in the future.

8.6 System Specification

A standard document created during the product creation is the system specification. It describes the system from the black box point of view: the **what** of the system. The system specification fits entirely in the *functional view*. Different acronyms and names are used. For example, in Philips one can find System Requirements Specification (SRS), but also system specification composed of smaller documents, for instance Functional Requirements Specification (FRS), while in ASML the name System Performance Specification (SPS) is used.

The system specification must cover multiple aspects:

- commercial, service and goods flow decompositions, see 8.6.1
- functions and features, see 8.6.2
- quantified requirements, such as performance, see 8.6.3
- external system interfaces, see 8.6.4
- standards compliance, see 8.6.5

8.6.1 Commercial, Service and Goods Flow Decomposition

The commercial granularity of sellable features and the allowed configurations can be visualized in a commercial configuration graph, as shown in Figure 8.10. All items in such a graph will appear in brochures, folders, and catalogues. Note that the commercial granularity is often somewhat coarser than the design decomposition. The commercial packaging is optimized to enable the sales process and to increase the margin. In some businesses the highest margin is in the add-ons, the accessories. In that case the add-ons are not part of the standard product to protect the margin.

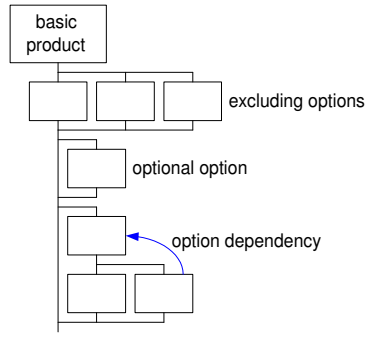


Figure 8.10: Commercial graph as means to describe commercial available variations and packaging

The commercial graph makes clear what the relations between commercial options are. Options might be exclusive (for example, either this printer or that printer can be connected, not both at the same time). Options can also be dependent on other options (for example, high definition video requires the memory extension to be present). The decomposition model chosen is a commercial decision, at least as long as the technical implications are feasible and acceptable in cost.

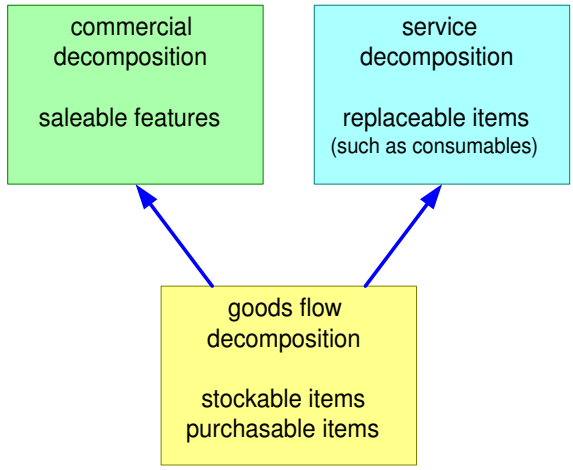


Figure 8.11: Logistic decompositions for a product

The same strategy can be used to define and visualize the decompositions needed for service (customer support, maintenance) and goods flow (ordering, storage and manufacturing of goods). Figure 8.11 shows the decompositions with their main decomposition drivers. These decompositions are not identical, but they are related. The goods flow decomposition must support the commercial as well as the service decomposition. The goods flow decomposition has a big impact on the cost side of the goods flow (goods=costs!) and must be sufficiently optimized for cost efficiency. The service decomposition is driven by the need to maintain systems efficiently, which often means that minimal parts should be replaced. The granularity of the service decomposition is finer than the commercial decomposition. The goods flow decomposition, which supports the commercial and the service decomposition, has a finer granularity than both these decompositions. At the input side is the goods flow decomposition determined by the granularity of the supply chain.

All three decompositions are logistics-oriented. These decompositions provide the structure, which is used in logistics information systems (MRP). In Philips the information in all three decompositions is stored in the so-called 12NC system, a logistics identification scheme deployed in the *Technical Product Documentation (TPD)*. The TPD is the formal output of the product creation process. The 12NC system defines conventions for decompositions and standardizes the identification of the components.

8.6.2 Function and Feature Specifications

The product specification defines the functions and features of the product. The decomposition for this description is again different from the commercial decomposition. The commercial decomposition is too coarse to use it as a basis for the product specification. The technical decomposition in functions and features is a building box to compose commercial products.

technical functions \ *products*	home cinema system	flat screen cinema TV	bedroom TV
HD display	+	+	-
SD->HD up conversion	+	+	-
HD->SD down conversion	+	+	o
HD storage	o	-	-
SD storage	o	-	o
HD IQ improvement	+	+	-
SD IQ improvement	+	+	+
HD digital input	+	+	o
SD digital input	+	+	o
SD analog input	o	+	+
6 HQ channel audio	+	o	-
2 channel audio	-	+	+

legend	
+	present
o	optional
-	absent

Figure 8.12: Mapping technical functions on products

Figure 8.12 shows a mapping of technical functions and features onto products. The technical functions and features should still be oriented towards the *what* of the product. In practice this view emerges slowly after many iterations between design decompositions and commercial, service and goods flow decompositions. This type of maps is used in several methods, for instance Quality Function Deployment (QFD) [67], or in PULSE [20].

8.6.3 Quantified requirements

The system requirements must be specified quantitatively and verifiable in the functional view, see for instance Gilb's recommendations in [26]. This holds for many requirements from performance and reliability to qualities that are more difficult to quantify such as extendibility.

Quantification can only take place in conjunction with the circumstances in which this quantification is valid. In easy cases a simple maximum value, which

is valid under all circumstances, is sufficient. In many systems quantification is more complicated: for instance the system performance depends on the user settings of the system.

In moderately complex systems it is sufficient to define a limited set of performance points in the parameter space. For more performance-critical and complex systems an external performance model might be required. This describes the required relation between performance and user settings. An example was shown in Figure 8.7, which shows a throughput model for a waferstepper, see Section 8.4.

8.6.4 External Interfaces

The external interfaces of the system can be described in a layered fashion, such as the OSI-ISO Model [81]. A large part of the interface descriptions will be covered by referring to standards, see Subsection 8.6.5.

The highest layers of the interface describe the semantics of the information. The syntax and representation aspects are described in the data model or data dictionary.

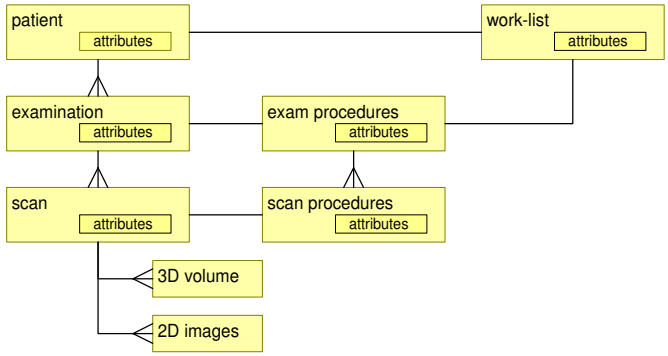


Figure 8.13: Example of a partial information model described by an entity relationship diagram

An information model in the *functional view* describes the information as seen from outside the system. It should not contain internal design choices. This information model is an important means to decouple interoperating systems. The functional behavior of the systems is predictable as long as all systems adhere to this information model. Figure 8.13 shows an example of a part of an information model, based on an entity relationship diagram [29]

The ingredients of an external information model are:

- entities
- relations between entities
- operations on entities

The most difficult part of the information model is to capture the semantics of the information. An information model defines the intended meaning of the information, in terms of entities, their meaning, the relation with other entities and possible operations that can be applied on these entities. Often other means are required as well such as an ontology, conventions for semantics, and formal notations.

The technical details of the information model, such as exact identifiers, data types and ranges are defined in the datamodel. The term data dictionary is also often used for this lower level of definitions.

8.6.5 Standards

Compliance with standards is part of the product specification. The level of compliance and possible exceptions need to be specified. Duplication of information in the standard must be avoided, because redundancy creates more maintenance work and increases the chance of inconsistencies in the specification. The nice characteristic of standards in general is that the standards are extensively described and well defined. An implementation that follows a standard is often straightforward engineering work, without the uncertainty of most other parts of the product specification.

Architecting work is, nevertheless, required in deciding on standards and in designing the implementation. Figure 8.14 shows the forces working upon the selection of standards. The market and business environment more or less dictate a set of standards. If the product does not comply with those standards the system is not viable. Some of these standards are mandatory due to legislation (for instance mandated by the VDE in Germany or the FDA in the United States), others are de facto musts (for instance DICOM, the medical imaging communication standard).

The use of the standard and the compliance level depend on the intended use. A key question for the architect is: *What is the intention of the standard?* Standards are created by domain experts. The domain experts make all kinds of conceptual assumptions. Using a standard in a way that does not correspond well with these assumptions, can create many specification and design problems. Good understanding of the underlying conceptual assumptions is a must for the architect.

The standard can have significant implementation consequences, for instance

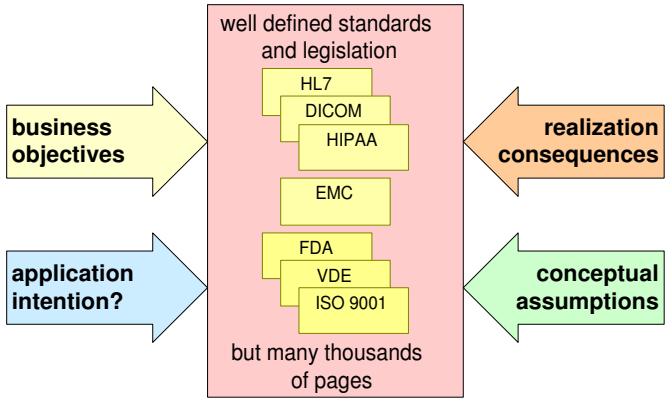


Figure 8.14: The standards compliance in the *functional view* in a broader force field.

in the amount of effort needed or the amount of license costs involved in creating the implementation. These costs must be balanced with the created customer value.

A major problem with standards compliance is the massive amount of documentation and know-how that is involved. The architect must find out the essence in terms of *objectives*, *intention*, *assumptions* and *consequences* of standards. In fact the architect must have a *CAFCR* mental model per standard[1]. For communication purposes the architect can make this model explicit.

[1] the CAFCR model describes in fact the architecture of the standard itself.

8.7 Overview of the Submethods in the CAF views

Figure 8.15 shows an overview of the submethods that are discussed in this chapter. These submethods are positioned in the *Customer Objectives View*, *Application View* and the *Functional View*. This positioning is not a black and white proposition, many submethods address aspects from multiple views. However, the positioning based on the essence of the submethod helps to select the proper submethod.

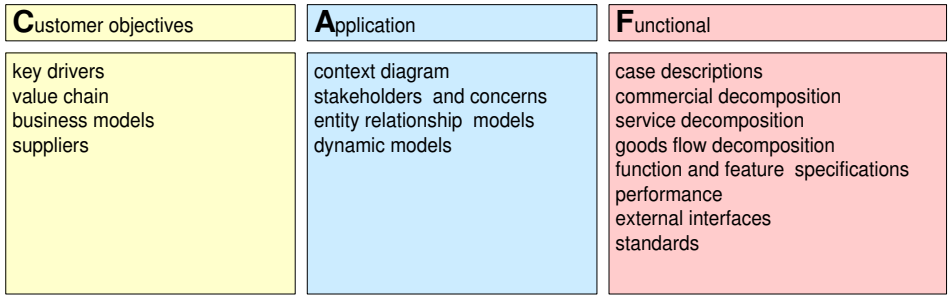


Figure 8.15: Overview of the submethods discussed in this chapter, positioned in the CAF views

Chapter 9

Submethods in the CR Views

9.1 Introduction

Decomposition is widely used in the conceptual and realization view. Section 9.2 describes a few *decompositions*, and introduces *interfaces*. From components to system qualities is more than a simple accumulation of component data. The system behavior and characteristics are described by *qualities* in Section 9.3. The conceptual and realization views also provide information to support the project manager. Section 9.4 describes some submethods to support project management.

9.2 Decomposition

Decomposition and modularity are well known concepts, which are the fundamentals of software engineering methods. A nice article by Parnas [61] discusses decomposition methods.

The decomposition can be done along different axes. Subsection 9.2.1 shows *construction* as axis, and Subsection 9.2.2 shows the *functional* decomposition. The decomposition into concurrent activities and the mapping on processes, threads and processors is called the execution architecture, which is described in Subsection 9.2.4.

The design of complex systems always requires multiple decompositions, for instance a construction and a functional decomposition. Subsection 9.2.3 describes a submethod to cope with multiple decompositions. The relations between the decompositions are described by mappings, described in Subsection 9.2.5.

Decompositions results in components. The interfacing between components is discussed in Subsection 9.2.6.

9.2.1 Construction Decomposition

The construction decomposition views the system from the construction point of view, see Figure 9.1 for an example. In this example the decomposition is structured to show layers and the degree of domain know-how. The vertical layering defines the dependencies: components in the higher layers depend on components in the lower layers. Components are not dependent on components at the same or higher layer. The amount of domain know how provides an indication of the added value of the components. More generic components are more likely to be shared in a broader application area, and are more likely to be purchased instead of being developed.

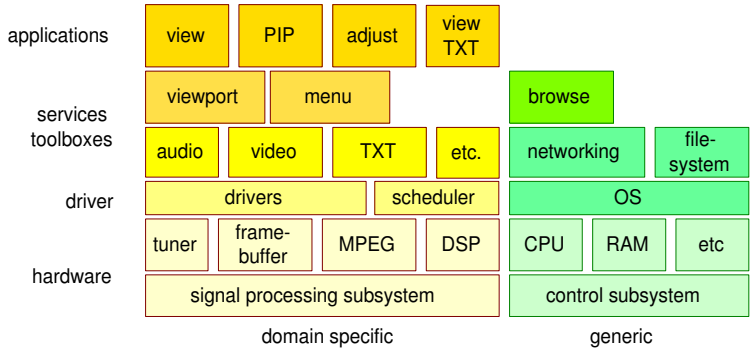


Figure 9.1: Example of a construction decomposition of a simple TV. The vertical axis is used for layers, where higher layers depend on lower layers, but not vice versa. In horizontal direction the left hand side shows the domain specific components, the right hand side shows the more generic components.

The construction decomposition is mostly used for the design management. It defines units of design, as these are created and stored in repositories and later updated. The atomic units are aggregated into compound design units. In software the compound design units are often called *packages*, in hardware they are called *modules*. The blocks in Figure 9.1 are at the level of these packages and modules. *Packages and modules* are used as unit for testing and release and they often coincide with organizational ownership and responsibility.

In hardware this is quite often a very natural decomposition, for instance into cabinets, racks, boards and finally integrated circuits, Intellectual property (IP) cores and cells. The components in the hardware are very tangible. The relationship with a number of other decompositions is reasonably one to one, for instance with the work breakdown for project management purposes.

The construction decomposition in software is more ambiguous. The structure of the code repository and the supporting build environment comes close to the hardware equivalent. Here files and packages are the aggregating construction levels. This decomposition is less tangible than the hardware decomposition and the relationship with other decompositions is sometimes more complex.

9.2.2 Functional Decomposition

The functional decomposition decomposes end user functions into more elementary functions. The elementary functions are internal, the decomposition in elementary functions is not easily observable from outside the system. In other words, the **what** is worked out in **how**. Be aware of the fact that the word *function* in system design is heavily overloaded. No attempt is made to define the functional decomposition more sharply, because a sharper definition does not provide more guidance to architects. Main criterium for a good functional decomposition is its useability for design. A functional decomposition provides insight how the system will accomplish its job. MASCOT [9] is an example of a method where a functional decomposition is based on data flow.

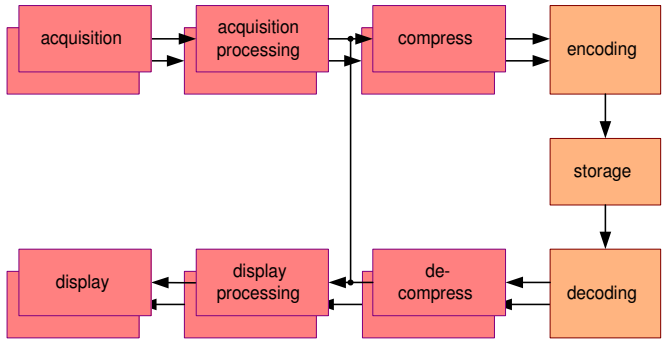


Figure 9.2: Example functional decomposition camera type device

Figure 9.2 shows an example of (part of) a functional decomposition for a camera type device. It shows a data flow with communication, processing, and

storage functions and their relations. This functional decomposition is **not** addressing the control aspects, which might be designed by means of a second functional decomposition, this time taken from the control point of view.

9.2.3 Designing with Multiple Decompositions

Most designers don't anticipate cross system design issues. During the preparation of design team meetings designers often do not succeed in submitting system level design issues. This limited anticipation is caused by the locality of the viewpoint, implicitly chosen by the designers. The designers are, while they working on a component, concerned about many design characteristics. Examples of design characteristics are *Signal to noise ratio (SNR)*, *accuracy*, *memory usage*, *processor load*, and *latency*.

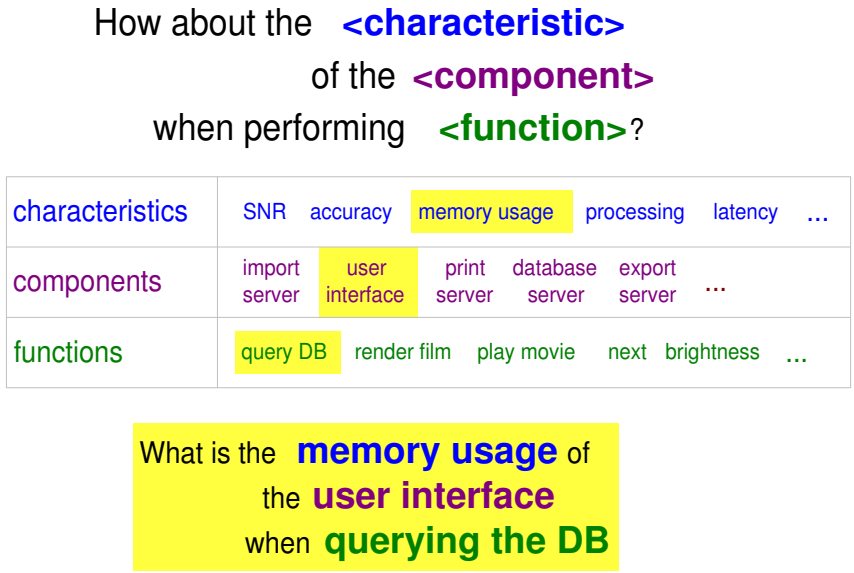


Figure 9.3: The question generator for multiple decompositions generates a question for every point in the Question space. The generic question is shown at the top. An example is shown below. The table shows a partial population for the three dimensions. The question at the bottom is generated by substituting one value from every row.

Figure 9.3 shows a method to help designers to find system design issues, based on the *Question space*. The question space is a three dimensional space.

Two dimensions are the decomposition dimensions (construction and functional); the last dimension is the design characteristic dimension. The design characteristics on this axis must be specific and quantifiable. A source of inspiration to find these characteristics are the qualities, described in Chapter 10, where the challenge is to find the specific and quantified characteristics that contribute to the quality.

For every point in this 3D space a question can be generated in the following way:
How about the <*characteristic*> of the <*component*> when performing <*function*>?
Which will result in questions like:
How about the *memory usage* of the *user interface* when *querying the database*?

The designers will not be able to answer most of these questions. Simply asking these questions helps the designer to change the viewpoint and discover many potential issues. Fortunately, most of the (not answered) questions turn out to be irrelevant. The answer to the memory usage question above might be *insignificant* or *small*. The more detailed memory usage questions are irrelevant as long as the total functionality fits in the available memory.

The architect can apply a priori know-how to select the most relevant questions in the 3D space, for instance:

Critical for system performance Every question that is directly related to critical aspects of the system performance is relevant. For example *What is the CPU load of the motion compensation function in the streaming subsystem?* will be relevant for resource constrained systems.

Risk planning wise Questions regarding critical planning issues are also relevant. For example *Will all concurrent streaming operations fit within the designed resources?* will greatly influence the planning if resources have to be added.

Least robust part of the design Some parts of the design are known to be rather sensitive, for instance the priority settings of threads. Satisfactory answers should be available, where a satisfactory answer might also be *we scheduled a priority tuning phase, with the following approach*.

Suspect parts of the design Other parts of the design may be suspect for a number of reasons. Experience, for instance, learns that response times and

throughput do not get the required attention of software designers (experience-based suspicion). Or we may have to allocated an engineer to the job with insufficient competence (person-based suspicion).

Some questions address a line or a plane in the multi dimensional space. An example of such an improved question is a memory budget for the system, thereby addressing all memory aspects for both functions and components in one budget.

9.2.4 Execution Architecture

The execution architecture is the run-time architecture of a system. The process[1] decomposition plays an important role in the execution architecture. Figure 9.4 shows an example of a process decomposition.

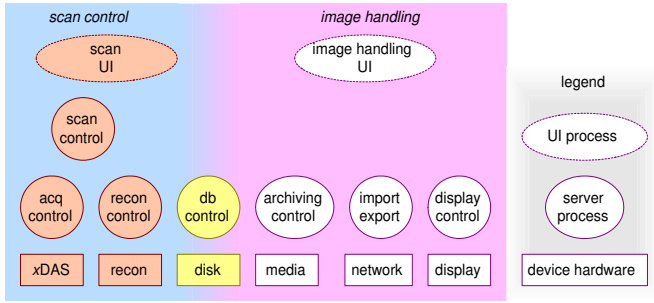


Figure 9.4: An example of a process decomposition of a MRI scanner.

One of the main concerns for process decomposition is concurrency: which concurrent activities are needed or running, and how do we synchronize these activities? Two techniques to support asynchronous functionality are widely used in operating systems: processes and threads. Processes are self sustained, which own their own resources, especially memory. Threads have less overhead than processes. Threads share resources, which makes them more mutually dependent. In other words processes provide better means for separation of concerns.

The execution architecture must map the functional decomposition on the process decomposition. This mapping must ensure that the timing behavior of the system is within specification. The most critical timing behavior is defined by the dead lines. Missing a dead line may result in loss of throughput or functionality.

[1] Process in terms of the operating system

The timing behavior is also determined by the choice of the synchronization methods, by the granularity of synchronization and by the scheduling behavior. The most common technique to control the scheduling behavior is by means of priorities. This requires, of course, that priorities are assigned. Subsystems with limited concurrency complexity may not even need multiple threads, but these subsystems can use a single thread that keeps repeating the same actions all the time. The mapping is further influenced by hardware software allocation choices, and by the construction decomposition. A well known method in the hard real time domain is DARTS (Design Approach for Real Time Systems) [27]. This methods provides guidelines to identify hard real time requirements, translate them in activities and to map activities on tasks. DARTS then describes how to design the scheduling priorities.

In practice many components from the construction decomposition are used in multiple functions, and are mapped on multiple processes. These shared components are aggregated in shared or dynamic-link libraries (dll's). Sharing the program code run-time is advantageous from memory consumption point of view.

9.2.5 Relations between Decompositions

The decompositions that are made as part of the design are related to each other. A mapping or allocation is required to relate a decomposition with another decomposition. For instance the functional decomposition can be mapped on the construction decomposition: functions are allocated to components in the construction decomposition. Another example is that functions are mapped on threads in the execution architecture.

The difficult aspect of these mappings is that in most systems $n : m$ mappings are needed. Every decompositions serves its own purposes, such as *construction* and *configuration management* in the construction decomposition, *performance* and *image quality* in the data flow functional decomposition, and *timing* and *concurrency* in the execution architecture. Each decomposition must clearly serve its intended purpose. On top of that a clear mapping strategy must be described to relate the decompositions.

9.2.6 Interfaces

The interfaces are the complement of the components in a decomposition. A lot of work on interface specifications has been done, for instance in KOALA [78]. KOALA adds the notion of *provides* and *requires* interfaces to formalize depen-

dency relations. A powerful decoupling step is the use of *protocols* as described by Jonkers [39]. Protocols according to [39] describe the functional and dynamic behavior of interfaces.

In Subsection 8.6.4 the interfaces were already discussed in the context of external interfaces in the functional view. The internal interface can be specified analogous to the external interfaces. Part of the internal interface is also specified by an internal information model, for instance modeled via entity relationship diagrams. The internal information model abstracts from the implementation, by modelling the data concepts, relationships and activities. The internal information model extends the external information model with data that are introduced as design concepts. It can, for instance, show caches, indices and other structures that are needed to achieve the required performance.

9.3 Quality Design Submethods

This section discusses submethods to achieve the objectives for some of the qualities that will be discussed in Chapter 10. *Performance* is discussed in Subsection 9.3.1. *Budgetting*, a submethod that can be used for several qualities, is described in Subsection 9.3.2. Submethods for *Safety*, *Reliability* and *Security* are discussed in Subsection 9.3.3. *Start up* and *Shutdown* are discussed in Subsection 9.3.4. Subsection 9.3.5 describes briefly submethods with respect to *Value* and *Cost*. Subsection 9.3.6 discusses granularity, an important the design consideration.

9.3.1 Performance Modeling

System performance is being tackled by using complementary models, such as visual models and analytical models. For instance, flow can be visualized by showing the order, inputs, outputs and the type of data, and flow performance can be described by means of a formula. In figure 9.5 the performance is modeled by a visual model at the top and an analytical model below. The analytical model is entirely parameterized, making it a generic model that describes the performance over the full potential range. For every function in the visual model the order of the algorithm is determined and the parameterization for the input and output data. The analytical model should be a manageable formula to provide insight in the performance behavior. In this example for MR reconstruction Fourier transforms are order $n * log(n)$, while the other computations are order n.

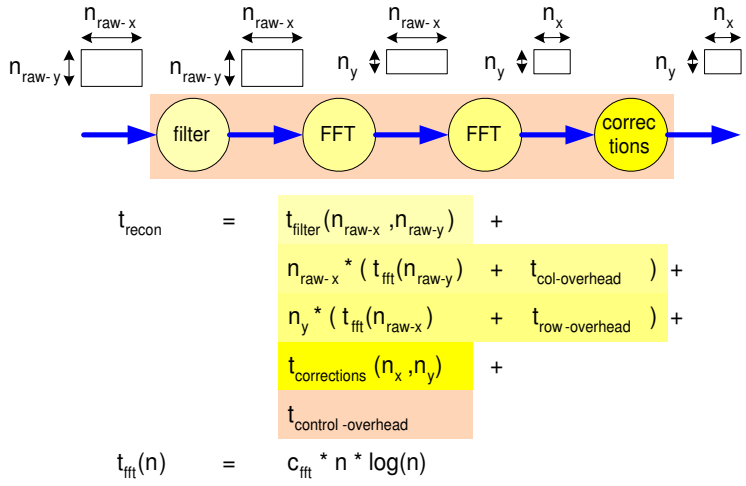


Figure 9.5: Flow model and analytical model of the image reconstruction in MR scanners. The analytical model is an algorithm to calculate the inherent computational costs.

The implementation of the system often reveals additional contributions to the processing time, resulting in an improved model, as shown in Figure 9.6. The pipeline model at the top of Figure 9.6, is extended with data transfer functions. The measurement below the model shows that a number of significant costs are involved in data transfer and control overhead. The original model of Figure 9.5 focuses on processing cost, including some processing related overhead. In product creation[2] the overhead plays a dominant role in the total system performance. Significant overhead costs are often present in initialization, I/O, synchronization, transfers, allocation and garbage collection (or freeing if explicitly managed).

Analytical performance models as shown in Figure 9.6 are powerful means to design, analyze and discuss performance. The difficulty in developing these models is in finding a manageable level of abstraction without losing too much predictive value. To develop the analytical model algorithmic analysis and empirical analysis need to be combined. It is my experience that analytical models with a manageable level of abstraction can be made for a wide variety of systems: MRI

[2]observed and coped with this problem in the following product developments: 1980 Video Display unit, 1981 Oncology Support, 1984 Digital Cardio Imaging, 1984 MRI user interface, 1986 MRI data acquisition, 1992 Medical Imaging workstation, 2002 Audio/Video processing.

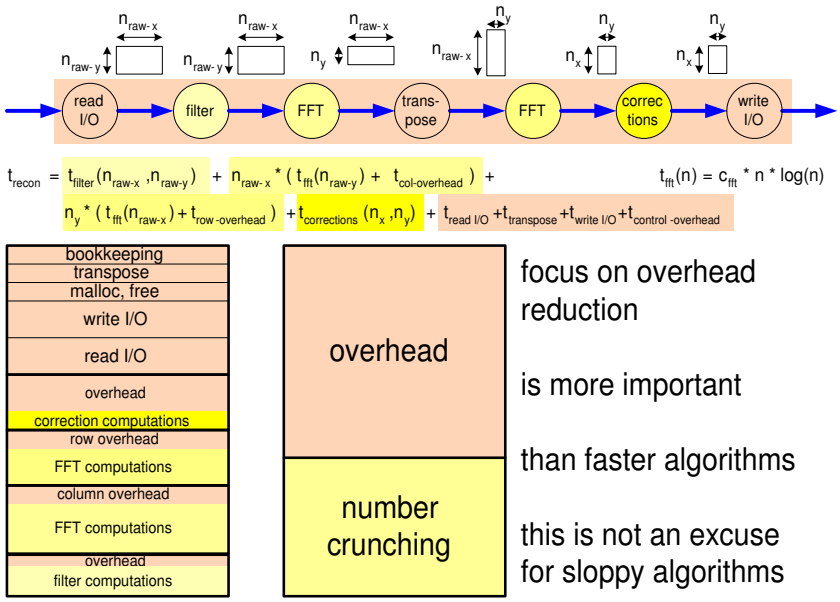


Figure 9.6: Example of performance analysis and evaluation. Implementation specific functions are added to the flow model and the analytical model. Below timing measurements are added, and classified as overhead and number crunching.

scanners, Digital Cardio Imaging, Medical Imaging Workstation, Wafersteppers, and Audio and Video processing systems.

The actual characteristics of the technology being used must be measured and understood in order to make a good (reliable, cost effective) design. The basic understanding of the technology is created by performing micro-benchmarks: measuring the elementary functions of the technology in isolation. Figure 9.7 lists a typical set of micro-benchmarks to be performed. The list shows infrequent and often slow operations and frequently applied operations that are often much faster. This classification implies already a design rule: slow operations should not be performed often[3].

The results of micro-benchmarks should be used with great care. The mea-

[3]This really sounds as an open door. However, I have seen many violations of this entirely trivial rule, such as setting up a connection for every message, performing I/O byte by byte et cetera. Sometimes such a violation is offset by other benefits, especially when a slow operation is in fact not very slow and when the brute force approach is both affordable as well as extremely simple.

	infrequent operations, often time-intensive	*often repeated operations*
database	start session finish session	perform transaction query
network, I/O	open connection close connection	transfer data
high level construction	component creation component destruction	method invocation same scope other context
low level construction	object creation object destruction	method invocation
basic programming	memory allocation memory free	function call loop overhead basic operations (add, mul, load, store)
OS	task, thread creation	task switch interrupt response
HW	power up, power down boot	cache flush low level data transfer

Figure 9.7: Typical micro-benchmarks for timing aspects

surements show the performance in totally unrealistic circumstances, in other words it is the best case performance. This best case performance is a good baseline to understand performance, but when using the numbers the real life interference (cache disturbance for instance) should be taken into account. Sometimes additional measurements are needed at a slightly higher level to calibrate the performance estimates.

The standard work about performance issues in computer architectures is the book by Hennesey and Patterson [30]. Here modelling and measurement methods can be found that can serve as inspiration for performance analysis of embedded systems.

9.3.2 Budgets

The implementation can be guided by making budgets for the most important resource constraints, such as memory size, response time, or positioning accuracy. The budget serves multiple purposes:

- to make the design explicit
- to provide a baseline to take decisions
- to specify the requirements for the detailed designs
- to have guidance during integration

- to provide a baseline for verification
- to manage the design margins explicitly

The simplification of the design into budgets introduces design constraints. Simple budgets are entirely static. If such a simplification is too constraining or costly then a dynamic budget can be made. A dynamic budget uses situational determined data to describe the budget in that situation. The architect must ensure the manageability of the budgets. A good budget has tens of quantities described. The danger of having a more detailed budget is the loss of overview.

step	example
1A measure old systems	micro-benchmarks, aggregated functions, applications
1B model the performance starting with old systems	flow model and analytical model
1C determine requirements for new system	response time or throughput
2 make a design for the new system	explore design space, estimate and simulate
3 make a budget for the new system:	models provide the structure measurements and estimates provide initial numbers specification provides bottom line
4 measure prototypes and new system	micro-benchmarks, aggregated functions, applications profiles, traces
5 Iterate steps 1B to 4	

Figure 9.8: Budget-based design flow

Figure 9.8 shows a budget-based design flow. The starting point of a budget is a model of the system, from the conceptual view. An existing system is used to get a first guidance to fill the budget. In general the budget of a new system is equal to the budget of the old system, with a number of explicit improvements. The improvements must be substantiated with design estimates and simulations of the new design. Of course the new budget must fulfill the specification of the new system, sufficient improvements must be designed to achieve the required improvement.

Early measurements in the integration are required to obtain feedback once the budget has been made. This feedback will result in design changes and could even result in specification changes.

9.3.3 Safety, Reliability and Security

The qualities *safety*, *reliability* and *security* share a number of concepts, for example:

- containment
- graceful degradation
- interlock, for instance dead man switch
- detection and tracing of failures,
- logging of operational data for post mortem analysis, e.g. flight recorder
- redundancy

All three qualities are covered by an extensive set of methods. Highly recommended is the work of Neumann [56]. A lot of literature is based on work in the aerospace industry. A good starting point to the literature is the home page of the International Council on Systems Engineering [36].

A common guideline in applying any of these concepts is that the more critical a function is, the higher the understandability should be, or in other words the simpler the applied concepts should be. Many elementary safety functions are implemented in hardware, avoiding large stacks of complex software.

Specialized engineering disciplines exist for Safety, Reliability and Security. These disciplines have developed their own methods. One class of methods relevant for system architects is the class of analysis methods that start with a (systematic) brainstorm, see figure 9.9. The Medical HACCP Alliance [75] provides extensive documentation for Hazard Analysis And Critical Control Point (HACCP) method for medical devices. A more systematic analysis provides input to improve the design.

Walk-through is another effective assessment method. A few use cases are taken and together with the engineers the implementation behavior is followed for these cases. The architect will especially assess the understandability and simplicity of the implementation. An implementation that is difficult to follow with respect to safety, security or reliability is suspect and at least requires more analysis.

9.3.4 Start up and Shutdown

The start up and the shutdown of the system are related to many components and functions of the system. One of the common patterns is the *run level* concept. The

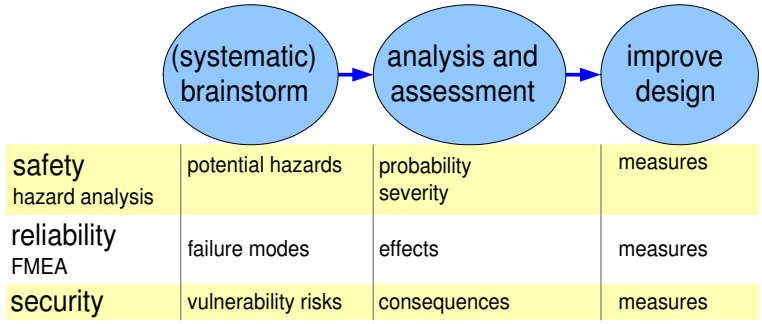


Figure 9.9: Analysis methods for safety, reliability and security

start up and shutdown are performed in phases, with increasing functionality and increasing integration, see for instance [49].

The current trend with more sophisticated power management, software downloading and roaming access networks increases the importance of clear design concepts to support these types of functionality.

9.3.5 Value and Cost

Many design decisions are made on an evaluation of the value of a design option versus the cost of this option. The production cost[4] of a system can be managed by making a decomposition of the cost and using the decomposition to create a cost budget.

Determination of the value of a design option is much more difficult. The value depends on the viewpoint. Some features are valuable for a particular stakeholder, for instance diagnostics for a service engineer and debugging for the developer. In general multiple viewpoints need to be somehow accumulated to create an *integral value*. QFD [67] uses multiple mappings with weight factors to "add" values together and create an integral value.

9.3.6 Granularity Determination

The granularity of operations is an important design choice. Fine granularity offers flexibility and fast response, at the cost of more overhead per operation.

[4]Within Philips the term Material and Labor Cost (MLC) is frequently used. The MLC determines the fixed cost of a product. The investment costs are variable costs. Different accounting practices are used to cope with the investment costs in the integral cost of the product.

Coarse granularity creates less overhead, at the cost of less flexibility and longer latencies. The determination of the granularity is an optimization problem that can be solved by applying optimization techniques from the operational research, see for an introduction [8].

Examples of operations where the unit size of operation has to be chosen are: buffering, synchronization, processing and input/output. In the case of video processing examples of operation sizes are: pixel, line and frame. In real video processing systems at this moment the unit size is typical a quarter of a video frame. The latency of the video chain is critical for two reasons: the time difference between audio and video must be small (lip-synchronization) and zapping must be fast. Smaller unit sizes create too much overhead, larger unit sizes create too much latency.

9.4 Project Management Support

The architect supports the project leader. Typical contributions of the architect are an initial *work breakdown* and an *integration plan*. Many more project management submethods exist, see for instance [28], but most of them are less relevant for the system architect.

A work breakdown is in fact again another decomposition, with a more organizational point of view [28]. The work in the different work packages should be cohesive internally, and should have low coupling with other work packages.

Figure 9.10 shows an example of a work breakdown. The entire project is broken down in a hierarchical fashion: project, segment, work package. In this example color coding is applied to show the technology involved and to show development work or purchasing work. Both types of work require domain know how, but different skills to do the job.

Schedules, work breakdown and many technical decompositions are heavily influenced by the integration plan. Integration is the effort of combining the components into a (sub)system, and to get the integrated (sub) system to work in the intended way. During the integration many specification and design inconsistencies, oversights, misunderstandings and mistakes are detected, analyzed and solved. Integration typically costs a lot of time and effort. The risk in a project is that the integration takes too much time and effort. Sufficient and regular attention for the integration viewpoint makes the risk better manageable.

Figure 9.11 shows an example of an integration plan. The systems that are used for the actual integration, the *integration vehicles*, are the limiting resource

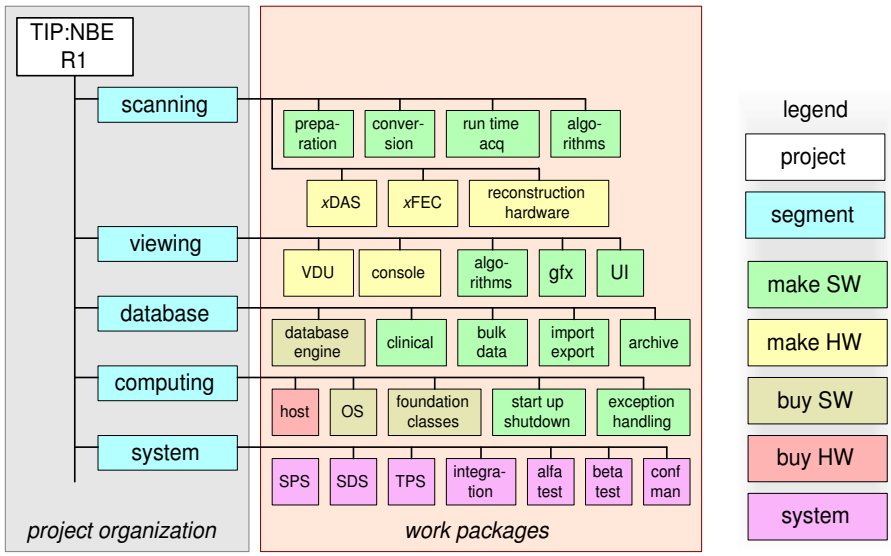


Figure 9.10: An example of a work breakdown from MRI scanner development. The project is organized in segments. The work in every segment is decomposed in work packages.

for integration. The integration plan is centered around 3 tiers of integration vehicles:

- partial systems to facilitate SW testing
- existing HW systems
- new HW systems

The partial systems for SW testing consist mostly of standard computer infrastructure. This computer infrastructure is very flexible and accessible from software point of view, but far from realistic from hardware viewpoint. The existing and new HW systems are much less accessible and more rigid, but close to the final product reality. The new HW system will be available late and hides many risks and uncertainties. The overall strategy is to move from good accessible systems with few uncertainties to less accessible systems with more uncertainties. A new application is first tested on a partial system for software testing. Then this application is tested on systems with existing hardware, with little hardware uncertainties. Finally this application is tested on the new base system. In general integration plans try to avoid stacking too many uncertainties by looking for

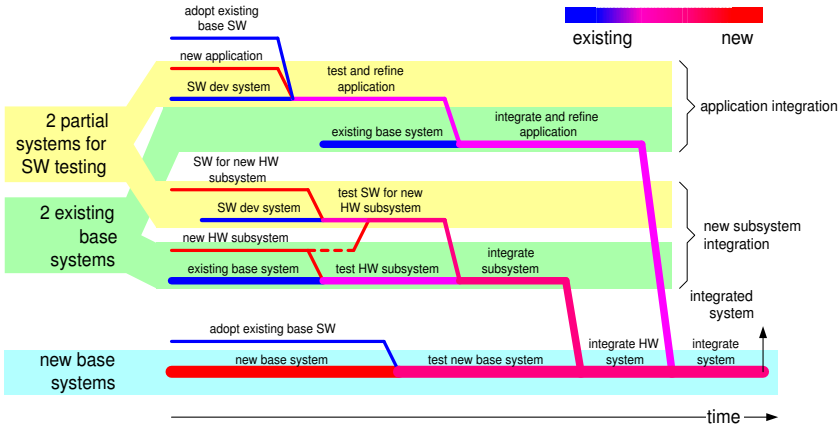


Figure 9.11: Example of an integration plan, with three tiers of integration vehicles. In this example two partial systems for software testing, two existing base systems and one new base system

ways to test new modules in a stable known environment, before confronting new modules with each other.

9.5 Overview of the Submethods in the CR views

Figure 9.12 shows an overview of the submethods that are discussed in this chapter. These submethods are positioned in the *Conceptual View* and the *Realization View*. This positioning is not a black and white proposition, many submethods address aspects from multiple views. However, the positioning based on the essence of the submethod helps to select the proper submethod.

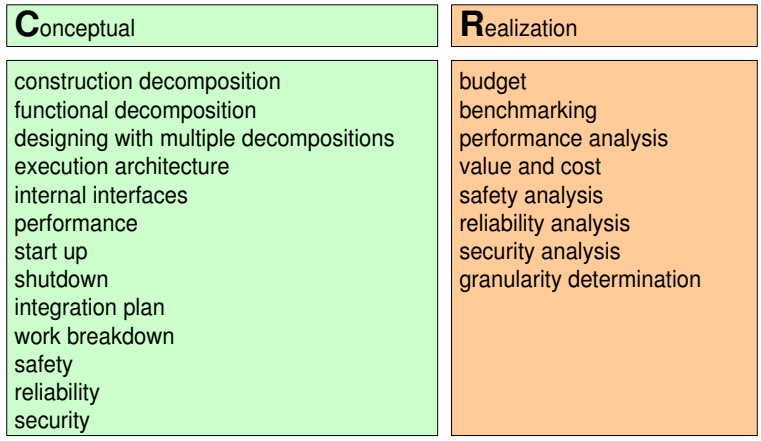


Figure 9.12: Overview of the submethods discussed in this chapter, positioned in the CR views

Chapter 10

Qualities as Integrating Needles

10.1 Introduction

The 5 CAFCR views become more useful when the information in one view is used in relation with neighboring views. One of the starting points is the use of the stakeholder concerns. Many stakeholder concerns are abstracted in a large set of more generic qualities. These qualities are meaningful in every view in their own way. Figure 10.1 shows the qualities as cross cutting needles through the CAFCR views.

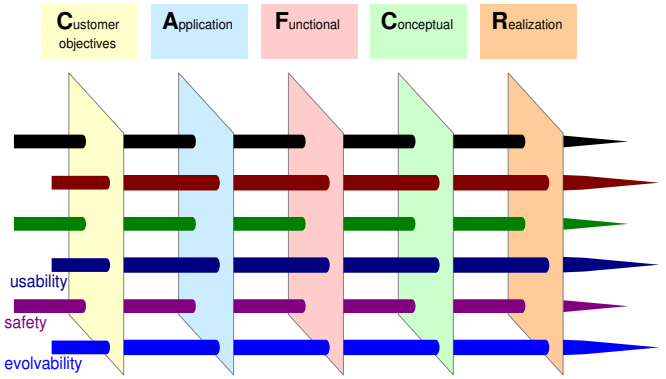


Figure 10.1: The quality needles are generic integrating concepts through the 5 CAFCR views

Section 10.2 shows an example of security as quality needle. In Section 10.3 a checklist of qualities is shown, with a definition of all qualities in the checklist.

10.2 Security as Example of a Quality Needle

As an example Figure 10.2 shows security issues for all the views. The green (upper) issues are the desired characteristics, specifications and mechanisms. The red issues are the threats to security. An excellent illustration of the security example can be found in [45].

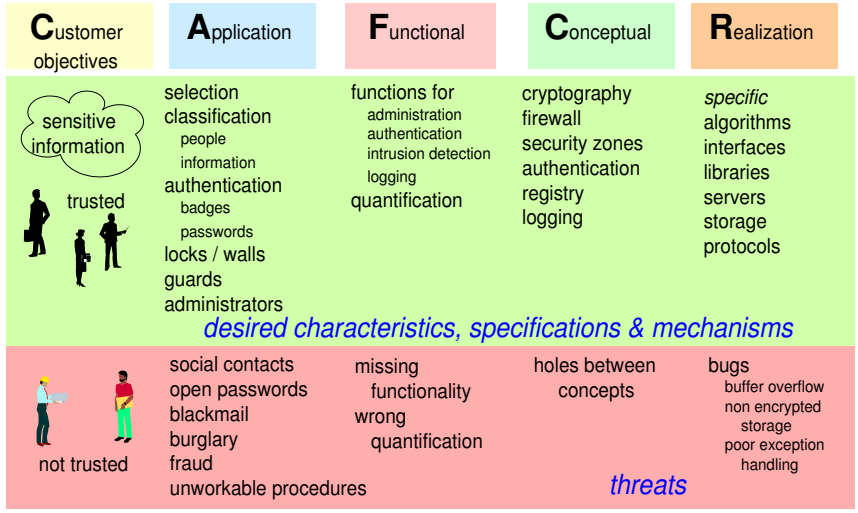


Figure 10.2: Example security through all views

10.2.1 Customer Objectives View

A typical customer objective with respect to security is to keep sensitive information secure, in other words only a limited set of trusted people has access. The other people (non trusted) should not be able to see (or worse, to alter) this information.

10.2.2 Application View

The customer will perform many activities to obtain security: from selecting trustful people to appointing special guards and administrators who deploy a security policy. Such a policy will involve classifying people with respect to their need for information and their trustfulness, as well as classifying information according to the level of security. To recognize *trusted* people authentication is required

by means of badges, passwords and in the future additional biometrics. Physical security by means of buildings, gates, locks, et cetera is also part of the security policy.

The security is threatened in many ways, from burglary to fraud, but also from simple issues like people forgetting their password and writing it on a yellow sticker. Social contacts of trusted people can unwillingly expose sensitive information, for instance when two managers are discussing business in a business lounge, while the competition is listening at the next table.

Unworkable procedures are a serious threat to security. For instance the forced change of passwords every month, resulting in many people writing down the password.

An interesting article is [15]. It shows how secret security procedures, in this case for passenger screening at airports, is vulnerable. It describes a method for terrorists how to reverse engineer the procedures empirically, which turns the effectiveness of the system from valuable to dangerous.

10.2.3 Functional View

The system under consideration will have to fit in the customer's security. Functions for authentication and administration are required. The performance of the system needs to be expressed explicitly. For instance the required confidence level of encryption and the speed of authentication have to be specified.

Security threats are usually caused by missing functionality or wrong quantification. This threat will surface in the actual use, where the users will find work arounds that compromise the security.

10.2.4 Conceptual View

Many technological concepts have been invented to make systems secure, for example cryptography, firewalls, security zones, authentication, registry, and logging. Every concept covers a limited set of aspects of security. For instance cryptography makes stored or transmitted data non-interpretable for non-trusted people.

Problems in the conceptual view are usually due to the non-ideal combination of concepts. For instance cryptography requires keys. Authentication is used to access and validate keys. The interface between cryptography and authentication is a risky issue. Another risky issue is the transfer of keys. All interfaces between the concepts are suspicious areas, where poor design easily threatens the security.

10.2.5 Realization View

The concepts are realized in hardware and software with specific mechanisms, such as encryption algorithms and tamper free interfaces. These mechanisms can be implemented in libraries, running at a distributed computer infrastructure. Every specific hardware and software element involved in the security concepts in itself must be secure, in order to have a secure system.

A secure realization is far from trivial. Nearly all systems have bugs. The encryption algorithm may be applicable, but if the library implementation is poor then the overall security is still poor. Well known security related bugs are buffer overflow bugs, that are exploited by hackers to gain access. Another example is storage of very critical security data, such as passwords and encryption keys, in non encrypted form. In general exception handling is a source of security threats in security.

10.2.6 Conclusion

Security is a quality that is heavily determined by the customer's way of working (application view). To enable a security policy of the customer a well-designed and well-implemented system is required with security functionality fitting in this policy.

In practice the security policy of customers is a large source of problems. Heavy security features in the system will never solve such a shortcoming. Another common source of security problems is poor design and implementation, causing a fair policy to be corrupted by the non-secure system.

Note that a very much simplified description of security has been presented, with the main purpose of illustration. A real security description will be more extensive than described here.

10.3 Qualities Checklist

Figure 10.3 shows a large set of qualities that can be used as a checklist for architecting. This set is classified to ease the access to the list. The qualities are not independent nor orthogonal, so every classification is at its best a means not a goal.

The following sections describe the different qualities briefly, in the *functional view*. Note that every quality can in general be described in each of the views. For instance, if the system is a head end system for a cable operator, then the

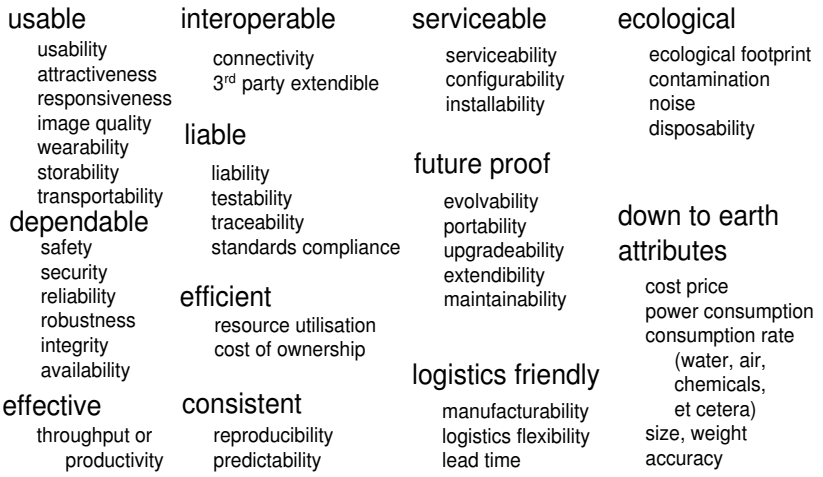


Figure 10.3: Checklist of qualities

useability of the (head end) system describes in the functional view the useability of the system itself, while in the customer objectives view the useability deals with the cable operator services.

The descriptions below are not intended to be **the** definition. Rather the list is intended to be used as a checklist, i.e. as a means to get a more all round view on the architecture.

10.3.1 Usable

useability The useability is a measure of usefulness and ease of use of a system.

attractiveness The appeal or attractiveness of the system.

responsiveness The speed of responding to inputs from outside.

image quality The quality of images (resolution, contrast, deformation, et cetera). This can be more generally used for output quality, so also sound quality for instance.

wearability The ease of wearing the system, or carrying the system around.

storability The ease of storing the system.

transportability The ease of transporting the system.

10.3.2 Dependable

safety The safety of the system. Note that this applies to all the stakeholders, for instance safety of the patient, operator, service employee, et cetera. Some people include the safety of the machine itself in this category. In my view this belongs to system reliability and robustness.

security The level of protection of the information in the system against unwanted access to the system.

reliability The probability that the systems operates reliable; the probability that the system is not broken and the software is not crashed. Here again the non-orthogonality of qualities is clear: an unreliable X-ray system is a safety risk when deployed for interventional surgery.

robustness The capability of the system to function in any (unforeseen) circumstances, including being foolproof for non-educated users.

integrity Does the system yield the *right* outputs.

availbility The availability of the system, often expressed in terms of (scheduled) uptime and the chance of unwanted downtime.

10.3.3 Effective

throughput or productivity The integral productivity level of the system. Often defined for a few use cases. Integral means here including aspects like start up shutdown, preventive maintenance, replacement of consumables et cetera. A bad attitude is to only specify the best case throughput, where all circumstances are ideal and even simple start up effects are ignored.

10.3.4 Interoperable

3^{rd} party extendable How open is the system for 3^{rd} party extensions? PCs are extremely open; many embedded systems are not extendable at all.

connectivity What other systems can be connected to the system and what applications are possible when connected?

10.3.5 Liable

liability The liability aspects with respect to the system; who is responsible for what, what are the legal liabilities, is the liability limited to an acceptable level?

testability The level of verifiability of the system, does the system perform as agreed upon?

traceability Is the operation of the system traceable? Traceability is required for determining liability aspects, but also for post mortem problem analysis.

standards compliance Large parts of the specification are defined in terms of compliance to standards.

10.3.6 Efficient

resource utilization The typical load of the system resources. Often specified for the same use cases as used for the productivity specification.

cost of ownership The cost of ownership is an integral estimate of all costs of owning and operating the system, including financing, personnel, maintenance, and consumables. Often only the sales price is taken as efficiency measure. This results in a suboptimal solution that minimize only the material cost.

10.3.7 Consistent

reproduceability Most systems are used highly repetitive. If the same operation is repeated over and over, the same result is expected all the time within the specified accuracy.

predictability The outcome of the system should be understandable for its users. Normally this means that the outcome should be predictable.

10.3.8 Serviceable

serviceability The ease of servicing the system: indication of consumable status, diagnostic capabilities in case of problems, accessibility of system internals, compatibility of replaceable units, et cetera.

configurability The ease of configuring (and maintaining, updating the configuration) the system

installability The ease of installing the system; for example the time, space and skills needed for installing.

10.3.9 Future Proof

evolvability The capability to change in (small) steps to adapt to new changing circumstances.

portability To be able to change the underlying platform, for instance from Windows NT to Linux, or from Windows 98SE to Windows XP.

upgradeability The capability of upgrading the entire or part of the system with improved features.

extendability The capability to add options or new features.

maintainability The capability of maintaining the well-being of the system, also under changing circumstances, such as end-of-life of parts or consumables, or new safety or security regulations.

10.3.10 Logistics Friendly

manufacturability The ease of manufacturing the system; for example time, space and skills needed for manufacturing.

logistics flexibility The capability to quickly adapt the logistics flow, for instance by fast ramp up (or down) supplier agreements, short lead times, low integration effort and second suppliers.

lead time The time between ordering the system and the actual delivery.

10.3.11 Ecological

ecological footprint The integral ecological load of the system, expressed in "original" ecological costs. This means that if electricity is used, the generation of electricity (and its inefficiency) is included in the footprint.

contamination The amount of contamination produced by the system

noise The (acoustical) noise produced by the system

disposability The way to get the system disposed, for instance the ability to decompose the system and to recycle the materials.

10.3.12 Down to Earth Attributes

These attributes (as the name indicates) are so trivial that no further description is given.

cost price

power consumption

consumption rate (water, air, chemicals, et cetera)

size, weight

accuracy

10.4 Summary

The qualities of a system can be generalized to the other CAFCR views. This generalization helps to understand the relationships between the views. Classification of the qualities is the basis for a checklist of qualities. This checklist is a tool for the architect: it helps the architect in determining the relevant qualities for the system to be created.

Chapter 11

Story Telling

11.1 Introduction

The CAFCR views and the quality needles are generic means to capture an architecture. The generic nature is powerful, however explorations in more depth are needed to understand the problem. Story telling followed by specific analysis and design work is a complementary method to do in depth exploration of parts of the specification and design. Starting a new product definition often derails in long discussions about generic specification and design issues. Due to lack of reality check these discussions can be very risky, and way too academic.

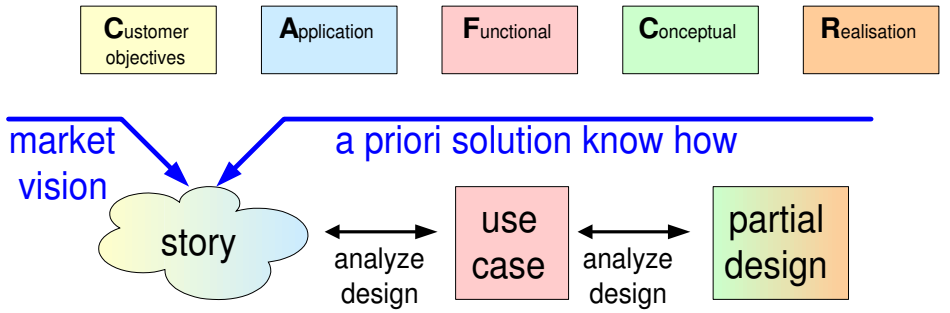


Figure 11.1: From story to design

The method provided here, based on story telling, is a powerful means to get the product definition quickly in a concrete factual discussion. The method is especially good in improving the communication between the different stakeholders.

This communication is tuned to the stakeholders involved in the different CAFCR views: the *story* and *use case* can be exchanged in ways that are understandable for both marketing-oriented people as well as for designers.

Figure 11.1 positions the story in the customer objectives view and application view. A good story combines a clear market vision with a priori realization know how. The story itself must be expressed entirely in customer terms, no solution jargon is allowed. The story is used to analyze specific parts of the specification: a use case. The use case is then used to explore specific parts of the design.

Section 11.2 describes how to create a story. The use of the story is explained in Section 11.3. The criteria for a good story are discussed in Section 11.4.

11.2 How to Create a Story?

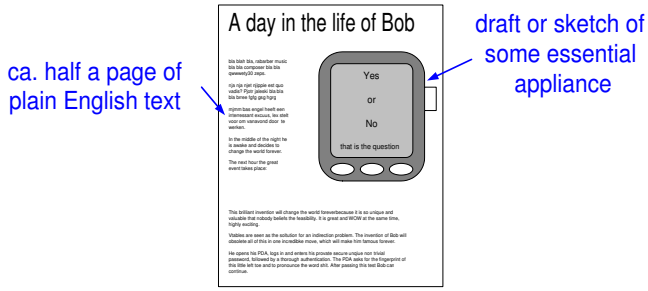


Figure 11.2: Example story layout

As shown in Figure 11.2 a story is a short single page story, preferably illustrated with sketches of the most relevant elements of the story, for instance the look and feel of the system being used. Other media such as cartoons, video or demonstrations using mockups can be used also. The *duration* or the *size* of the "story" must be limited to enable focus on the essentials.

Every story has a *purpose*, something the design team wants to learn or explore. The purpose of the story is often in the conceptual and realization views. The *scope* of the story must be chosen carefully. A wide scope is good to understand a wide context, but leaves many details unexplored. A useful approach is to use *recursively refined* stories: an overall story setting the context and a few other stories zooming in on aspects of the overall story.

The story can be written from several *stakeholder viewpoints*. The viewpoints should be carefully chosen. Note that the story is also an important means of

communication with customers, marketing managers and other domain experts. Some of the stakeholder viewpoints are especially useful in this communication.

11.3 How to Use a Story?

The story itself must be very accessible for all stakeholders. The story must be attractive and appealing to facilitate communication and discussion between those stakeholders. The story is also used as input for a more systematic analysis of the product specification in the functional view. All functions, performance figures and quality attributes are extracted from the story. The analysis results are used to explore the design options.

Normally several iterations will take place between story, case and design exploration. During the first iteration many questions will be raised in the case analysis and design, which are caused by the story being insufficiently specific. This needs to be addressed by making the story more explicit. Care should be taken that the story stays in the Customers views and that the story is not extended too much. The story should be sharpened, in other words made more explicit, to answer the questions.

After a few iterations a clear integral overview and understanding emerges for this very specific story. This insight is used as a starting point to create a more complete specification and design.

11.4 Criteria

Figure 11.3 shows the criteria for a good story. It is recommended to assess a story against this checklist and either improve the story such that it meets all the criteria or reject the story. Fulfillment of these criteria helps to obtain a useful story. The set of five criteria is a necessary but not sufficient set of criteria. The value of a story can only be measured in retrospect by determining the contribution of the story to the specification and design process.

Subsections 11.4.1 to 11.4.5 describe every criterion in more detail.

11.4.1 Accessible, Understandable

The main function of a story is to make the opportunity or problem communicable with all the stakeholders. This means that the story must be accessible and understandable for all stakeholders. The description or presentation should be such that

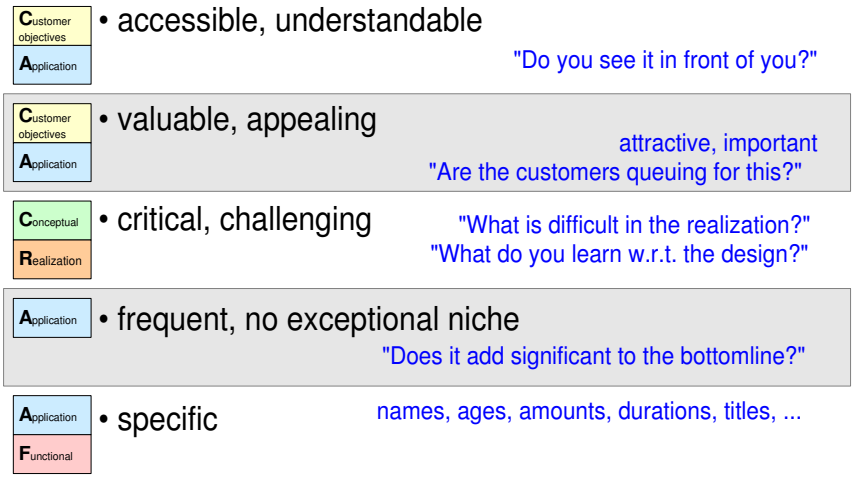


Figure 11.3: Criteria for a good story

all stakeholders can *live through*, *experience* or *imagine* the story. A "good" story is not a sheet of paper, it is a living story.

11.4.2 Valuable, Appealing

The opportunity or problem (idea, product, function, or feature) must be significant for the target customers. This means that it should be important for them, or valuable; it should be appealing and attractive.

Most stories fail on this criterium. Some so-so opportunity (whistle and bell-type) is used, where nobody gets really enthusiastic. If this is the case more creativity is required to change the story to a useful level of importance.

11.4.3 Critical, Challenging

The purpose of the story is to learn, define, and analyze new products or features. If the implementation of a story is trivial, nothing will be learned. If all other criteria are met and no product exists yet, then just do it, because it is clearly a quick win!

If the implementation is challenging, then the story is a good vehicle to study the trade-offs and choices to be made.

11.4.4 Frequent, no Exceptional Niche

Especially in the early exploration it is important to focus on the main line, the *typical* case. Later in the system design more specialized cases will be needed to analyze for instance more exceptional worst case situations.

A *typical* case is characterized by being frequent, it should not be an exceptional niche.

11.4.5 Specific

The value of a story is the specificity. Most system descriptions are very generic and therefore very powerful, but at the same time very non-specific. A good story provides focus on a single story, one occasion only. In other words, the thread of the story should be very specific.

A common pitfall for story writers is to show all possibilities in one story. For example one paragraph that describes all the potential goodies. Simply leave out such a paragraph, it only degrades the focus and value of the story.

A good story is in **all** aspects as specific as possible, which means that:

- persons playing a role in the story preferably have a name, age, and other relevant attributes
- the time and location are specific (if relevant)
- the content is specific (for instance is listening for **2 hours** to songs of **the Beatles**)

This kind of specific data is often needed to assess the other criteria, to bring it more alive, and in further analysis. If during the use of the story numbers have to be “invented”, it is recommended to improve the story by adding specific facts to the story.

11.5 Summary

Story telling is a means to become specific and concrete in the early product creation phases. Five criteria are described to create and to assess stories: accessibility, value, challenge, frequency and specificity.

Unfortunately, the research in this area took place many years after the case study in Part III. Some comparable effort in the case will be discussed in Chapter 16. In Part IV some more evidence from a different context will be provided for the story telling submethod.

Chapter 12

Threads of Reasoning

12.1 Introduction

The submethods provide generic means to cope with a limited part of the system architecture. The CAFCR model and the qualities provide a framework to position these results. The story telling is a means to do analysis and design work on the basis of concrete and specific facts. In this chapter a reasoning method is discussed to integrate all previous submethods. This reasoning method covers both the high level and the detailed views and covers the relation between multiple submethods and multiple qualities. The method is based on the identification of the points of tension in the problem and potential solutions.

The reasoning approach is explained as a 5 step approach. Section 12.2 provides an overview of the approach and gives a short introduction to each step. Section 12.3 describes the actual reasoning over multiple viewpoints: how to maintain focus and overview in such a multi-dimensional space? How to communicate and document? Section 12.4 explains how the threads of reasoning fit in the complete method.

12.2 Overview of Reasoning Approach

Fast exploration of the problem and solution space improves the quality of the specification and design decisions, as explained in Chapter 7. It is essential to realize that such an exploration is highly concurrent, it is neither top-down, nor bottom-up, see viewpoint hopping and decision making in Sections 7.2 and 7.8. In practice many designers find it difficult to make a start. In fact this does not have

to be difficult: most starting points can be used, as long as the method is used with a sufficient open mind (that means that the starting point can be changed, when the team discovers that more important specification or design decisions are needed).

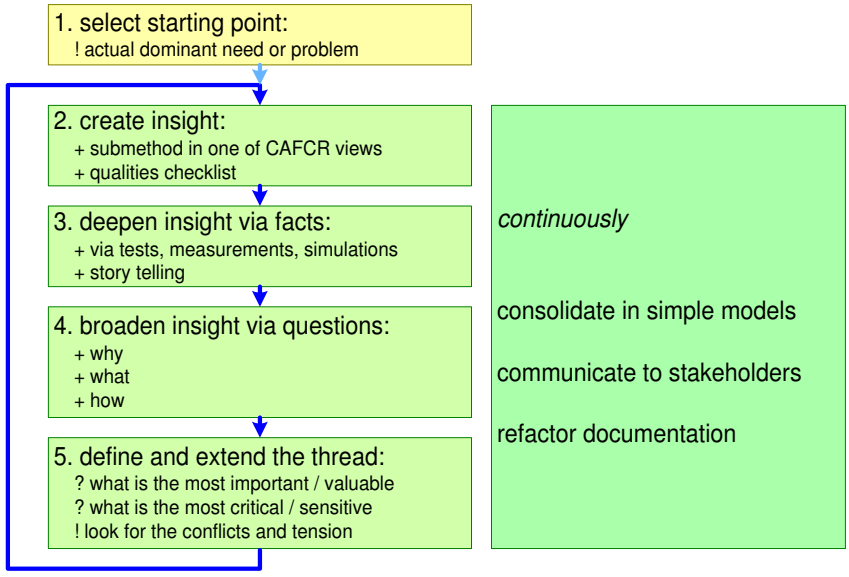


Figure 12.1: Overview of reasoning approach

Figure 12.1 shows an overview of the entire reasoning approach. Step 1 is to *select a starting point*. After step 1 the iteration starts with step 2 *create insight*. Step 3 is *deepening the insight* and step 4 is *broadening the insight* with the questions. The next iteration is prepared by step 5 *refining or selecting the next need or problem*.

During this iteration continuous effort is required to *communicate with the stakeholders* to keep them up to date, to *consolidate in simple models* that are used during analysis and discussions and to *refactor the documentation* to keep it up to date with the insights obtained.

12.2.1 Selecting a Starting Point

As stated earlier it is more important to get started with the iteration than to spend a lot of time trying to find the most ideal starting point. A very useful starting point is to take a need or problem that is very hot at the moment. If this issue

turns out to be important and critical then it needs to be addressed anyway. If it turns out to be not that important, then the outcome of the first iteration serves to diminish the worries in the organization, enabling it to focus on the important issues.

In practice there are many hot issues that after some iterations turn out to be non-issues. This is often causes by non-rational fears, uncertainty, doubt, rumors, lack of facts et cetera. Going through the iteration, which includes fact finding, quickly positions the issues. This is of great benefit to the organization as a whole.

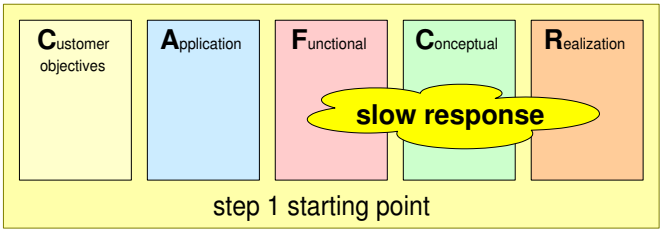


Figure 12.2: Example of a starting point: a slow system response discussed from the designer's viewpoint

The actual dominant needs or problems can be found by listening to what is mentioned with the greatest loudness, or which items dominate in all discussions and meetings. Figure 12.2 shows the response time as starting point for the iteration. This starting point was triggered by many design discussions about the cause of a slow system response and about potential concepts to solve this problem.

12.2.2 Building up Insight

The selected issue can be modeled by means of one of the many submethods as described in the CAFCR chapters. Doing this, it will quickly become clear what is known (and can be consolidated and communicated) and what is unknown, and what needs more study and is hence input for the next step. Figure 12.3 shows the *response time model* as potential submethod.

An alternative approach is to look at the issue from the perspective of quality. One then has to identify the most relevant qualities, by means of the checklist in Figure 10.3. These qualities can be used to sharpen the problem statement. Figure 12.3 shows the *performance* as quality to be used to understand the response time issue.

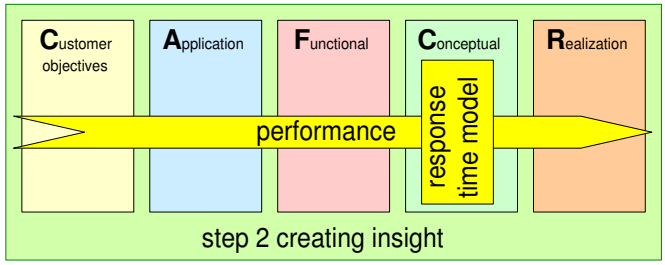


Figure 12.3: Example of creating insight: to study the required performance a response model of the system is made

12.2.3 Deepening the Insight

The insight is deepened by gathering specific facts. This can be done by simulations, or by tests and measurements on existing systems. At the customer side story telling helps to get the needs sufficiently specific, as illustrated by Figure 12.4.

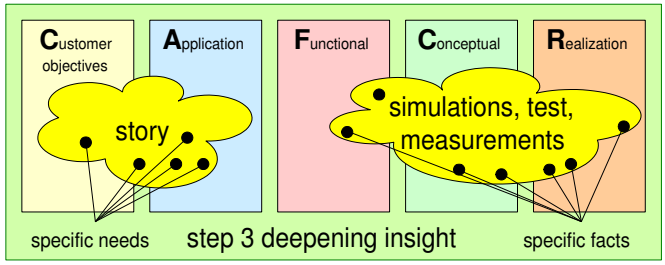


Figure 12.4: Deepening the insight by articulating specific needs and gathering specific facts by simulations, tests and simulations

It is important in this phase to *sample* specific facts and not to try to be complete. A very small subset of specific facts can already provide lots of insight. The speed of iteration is much more important than the completeness of the facts. Be aware that the iteration will quickly zoom in on the core design problems, which will result in sufficient coverage of the issues anyway.

12.2.4 Broadening the Insight

Needs and problems are never nicely isolated from the context. In many cases the reason why something is called a problem is because of the interaction between the function and the context. The insight is broadened by relating the need or problem to the other views in the CAFCR model. This can be achieved by the *why*, *what* and *how* questions as described in Section 7.7 and shown in Figure 12.5.

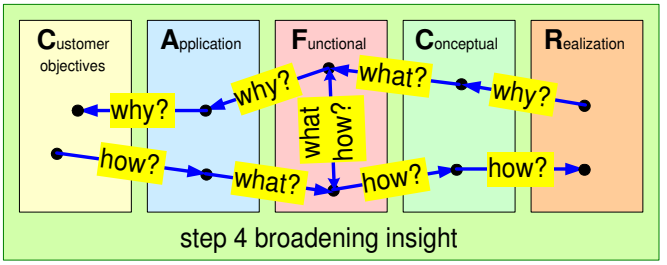


Figure 12.5: Broadening the insight by repeating why, what and how questions

The insight in the quality dimension can also be broadened by looking at the interaction with related qualities: what happens with safety, when we increase the performance?

12.2.5 Define and Extend the Thread

During the study and discussion of the needs and problems many new questions and problems pop up. A single problem can trigger an avalanche of new problems. Key in the approach is not to drown in this infinite ocean full of issues, by maintaining focus on important and critical issues. The most progress can be made by identifying the specification and design decisions that seem to be the most conflicting, i.e. where the most tension exists between the issues.

The relevance of a problem is determined by the *value* or the *importance* of the problem for the customer. The relevance is also determined by how challenging a problem is to solve. Problems that can be solved in a trivial way should immediately be solved. The approach as described is useful for problems that require some critical technical implementation. The implementation can be critical because it is difficult to realize, or because the design is rather sensitive[1] or rather

[1] for instance in MRI systems the radius of the gradient coil system and the cost price were related with $(r_{magnet} - r_{gradientcoil})^5$. 1 cm more patient space would increase the cost dramatically, while at the same time patient space is crucial because of claustrophobia.

vulnerable (for example, hard real-time systems with processor loads up to 70%).

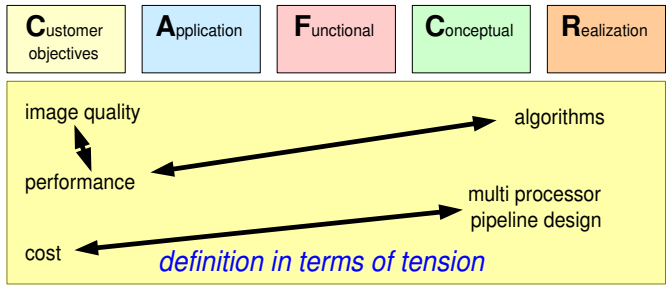


Figure 12.6: Example definition of the thread in terms of tension for a digital TV

Figure 12.6 shows the next crucial element to define the thread: identification the tension between needs and implementation options. The problem can be formulated in terms of this tension. A clearly articulated problem is half of the solution.

The example in Figure 12.6 shows the tension between the customer objectives and the design options. The image quality objective requires good algorithms that require a lot of processing power. Insufficient processing power lowers the system performance. The processing power is achieved by a pipeline of multiple processors. The cost of the number crunching capacity easily exceeds the cost target.

12.3 Reasoning

The reasoning by the architect is based on a combination of subjective intuition and objective analysis. The intuition is used to determine the direction and to evaluate results. The analysis partially validates the direction and partially helps the architect to develop his intuition further.

The assessment of the solutions is done by means of criteria. An objective ranking of the solutions can be made based on these criteria. The architect (and the other stakeholders) have their own subjective ranking based on intuition. By comparing the objective and subjective rankings a better understanding is achieved of both problem and solutions. This is shown in Figure 12.7. The increased understanding of the problem is used to improve the criteria. The increased understanding of the problem and the solutions influences the intuition of the architect (for instance this type of function is more expensive than expected). The increased

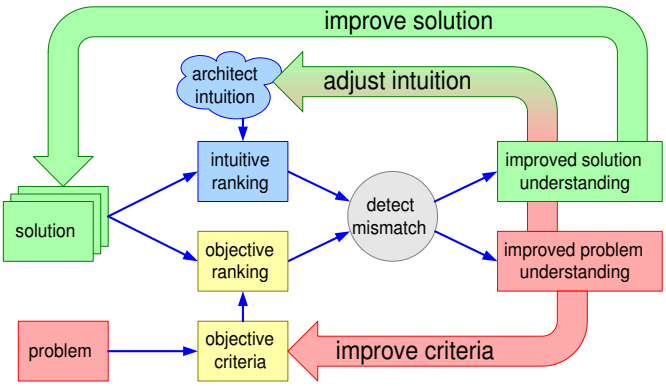


Figure 12.7: Reasoning as a feedback loop that combines intuition and analysis

understanding of the solution will trigger new solution(s).

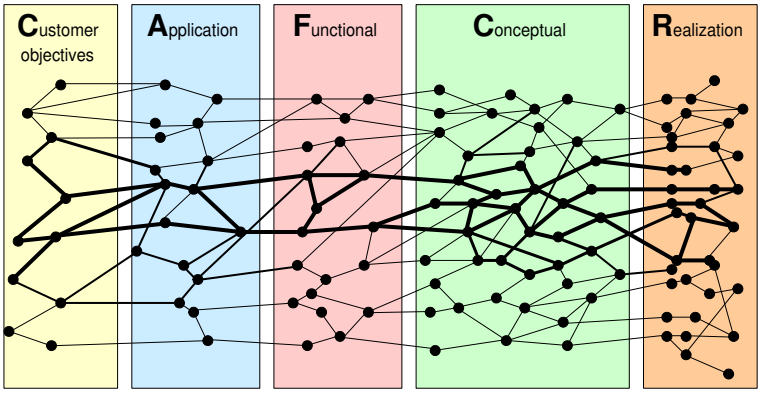


Figure 12.8: One *thread of reasoning* showing related issues. The line thickness is an indication for the weight of the relation.

During the reasoning a network of related issues emerges, as shown in Figure 12.8. Figure 12.8 visualizes the network as a graph, where a dot represents a specification or a design decision and a line represents a relation. Such a relation can be: *is implemented by*, *is detailed by*, *is conflicting with*, *enables or supports* et cetera. The thickness of the line indicates the weight of the relation (thin is weak, thick is strong).

This graph is a visualization of the thread of reasoning followed by an architect. Crucial in such a thread is that it is sufficiently limited to maintain overview

and to enable discussion and reasoning. A good thread of reasoning addresses relevant problem(s), without drowning in the real world complexity.

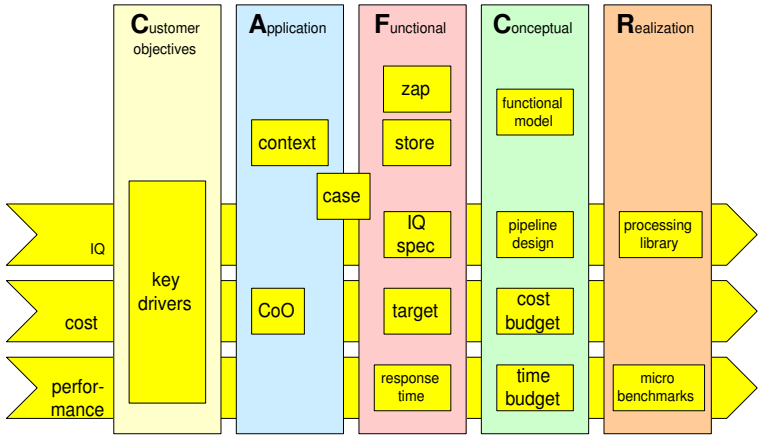


Figure 12.9: Example of the documentation and communication for a digital TV. The thread is documented in a structured way, despite the chaotic creation path. This structure emerges after several iterations.

A continuous concern is to communicate with the stakeholders and to consolidate the findings, for instance in documentation. Figure 12.9 shows the more structured way to document and communicate these findings. The architect needs several iterations to recognize the structure in the seeming chaotic thread of reasoning. This example discusses the thread that has been shown in Figure 12.6. This single thread of reasoning addresses three key drivers as shown in Figure 12.9: *IQ (Image Quality)*, *cost* and *performance*. Most information in the thread of reasoning addresses these key drivers, however some additional information emerges too, such as the *context* of the digital TV at home, the functionality of the *zap* and *store* functions and the internal *functional models*.

12.4 Outline of the complete method

The *threads of reasoning* are the integration means of the overall method. In this section a short description is given how the *threads of reasoning* are are combined with the *submethods*, *quality checklists* and *story telling* to form a complete method. The steps in the description refer to Figure 12.7. Note that this aspect is speculative, because it has not been applied and therefore cannot be evaluated at

this moment. Only an outline can be given now. A more detailed description of the method has to wait until further research is due.

The starting point (step 1) of a product creation is often a limited product specification, belonging in the *Functional view*. The next step is to explore (step 2) the customer context (*Customer Objectives* and *Application* views) and to explore the technical merits (*Conceptual* and *Realization* views). This exploration is used to identify a first set of customer-side opportunities and to identify the biggest technical challenges. During the exploration the *submethods* and *quality checklists* are used as a source of inspiration, for instance to determine the opportunity in the business model of the customer. Next (step 3) a *story* must be created that addresses the most important and valuable opportunities and the biggest technical challenges. The *story* is used to derive a first *use case* and to do a more thorough exploration (step 4) of the specification and the design. At this moment the first thread of reasoning is already visible (step 5), connecting a coarse product specification with customer opportunities and technical challenges. From this moment onwards the steps are repeated over and over, extending the thread of reasoning and creating one or two more threads of reasoning if needed. The submethods and the qualities are used during these iterations as a toolbox to describe specific parts of this creation process.

12.5 Summary

The reasoning approach is a means to integrate the CAFCR views and the qualities to design a system that fits entirely in the customer needs. The *threads of reasoning* approach is described by five steps. The result can be visualized as a graph of many related customer needs, specification issues and design issues. In this graph the core reasoning can be indicated around a limited set of key drivers or quality needs. In Chapter 17 the graph will be visualized for the Medical Imaging Workstation case.

Part III

Medical Imaging Case Description

Chapters in Part III:

Chapter 13

Medical Imaging in Chronological Order

13.1 Project Context

Philips Medical Systems is a very old company, dating back to 1896 when the first X-ray tubes were manufactured. Many imaging modalities have been added to the portfolio later, such as Ultra Sound, Nuclear Medicaid, Computed Tomography and Magnetic Resonance Imaging. Since the late seventies the management was concerned by the growing effort to develop the viewing functionality of these systems. Many attempts have been made to create a shared implementation of the viewing functionality, with failures and partial successes.

In 1987 a new attempt was started by composing a team, that had the charter to create a *Common Viewing* platform to be used in all the modalities. This team had the vision that a well designed set of SW components running on standard workstation hardware would be the solution. In the beginning of 1991 many components had been built. For demonstration purposes a *Basic Application* was developed. The *Basic Application* makes all lower level functionality available via a rather technology-oriented graphical user interface. The case description starts at this moment, when the *Basic Application* is shown to stakeholders within Philips Medical Systems.

13.2 Introduction

The context of the first release of Medical Imaging is shown in Section 13.1. The chronological development of the first release of the medical imaging workstation is described in Section 13.3. Sections 13.4 and 13.5 zoom in on two specific problems encountered during this period.

13.3 Development of Easyvision RF

The new marketing manager of the *Common Viewing* group was impressed by the functionality and performance of the *Basic Application*. He thought that a stand alone product derived from the *Basic Application* would create a business opportunity. The derived product was called Easyvision, the first release of the product was called Easyvision R/F. This first release would serve the URF X-ray market. The *Common Viewing* management team decided to create Easyvision RF in the beginning of 1991.

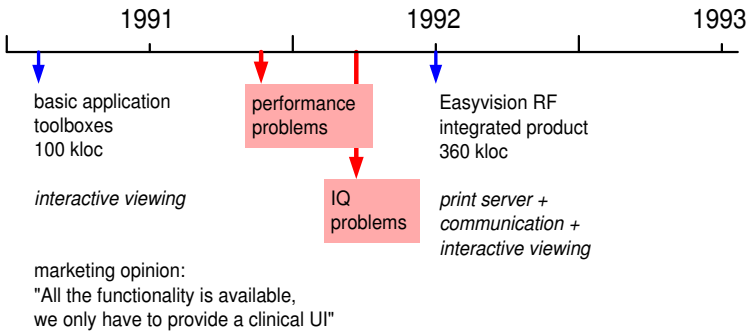


Figure 13.1: Chronological overview of the development of the first release of the Easyvision

The enthusiasm of the marketing people for the *Basic Application* was based on the wealth of functionality that was shown. It provided all necessary viewing functions and even more. Figure 13.1 shows the chronology, and the initial marketing opinion. Marketing also remarked: ”Normally we have to beg for more functionality, but now we have the luxury to throw out functionality”. The addition of viewing software to the conventional modality products[1] was difficult

[1] *Modality products* are products that use one imaging technique such as Ultra Sound, X-ray or

for many reasons, such as *legacy code and architecture*, and *safety and related testing requirements*. The Easyvision did not suffer from the legacy, and the self sustained product provided a good means to separate the modality concerns from the image handling concerns.

This perception of a nearly finished product, which only needed some user interface tuning and some functionality reduction, proved to be a severe underestimation. The amount of code in the 1991 *Basic Application* was about 100 kloc (kloc = thousand lines of code, including comments and empty lines), while the product contained about 360 kloc.

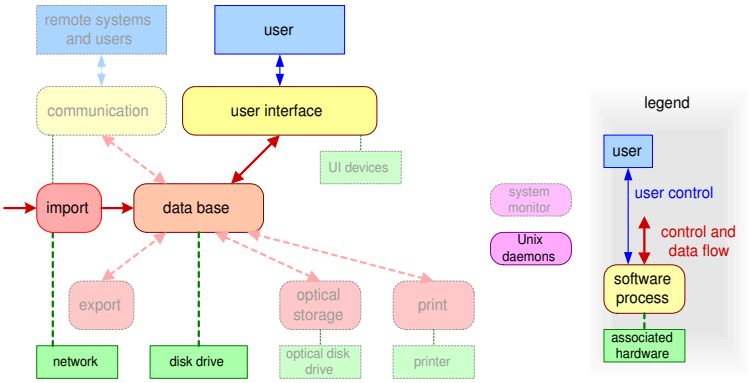


Figure 13.2: The functionality present in the Basic Application shown in the process decomposition. The light colored processes were added to create the Easyvision

The *Basic Application* provided a lot of viewing functionality, but the Easyvision as a product required much more functionality. The required additional functionality was needed to fit the product in the clinical context, such as:

- interfacing with modalities, including remote operation from the modality system
- storage on optical discs
- printing on film

Figure 13.2 shows in the process decomposition what was present and what was missing in the 1991 code. From this process decomposition it is clear that many more systems and devices had to be interfaced. Figures 13.2 and 13.3 are explained further in Chapter 15.

Magnetic Resonance Imaging

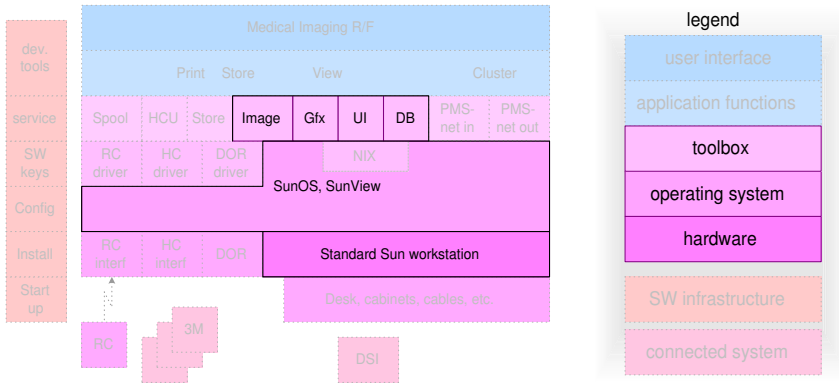


Figure 13.3: The functionality present in the Basic Application shown in the construction decomposition. The light colored components were added to create the Easyvision

Figure 13.3 also shows what was present and what was missing in the Basic Application, but now in the construction decomposition. Here it becomes clear that also the application-oriented functionality was missing. The *Basic Application* offered generic viewing functionality, exposing all functionality in a rather technical way to the user. The clinical RF user expects a very specific viewing interaction, that is based on knowledge of the RF application domain.

The project phases from the conception of a new product to the introduction in the market is characterized by many architectural decisions. Architecting methods are valuable means in this period. Characteristic for an immature architecting process is that several crises occur in the integration. As shown in Figure 13.1 both a performance and a (image quality related) safety crisis happened in that period.

13.4 Performance Problem

The performance of the system at the end of 1991 was poor, below expectation. One of the causes was the extensive use of memory. Figure 13.4 shows the performance of the system as a function of the memory used. It is also indicates that a typically loaded system at that moment used about 200 MByte. Systems which use much more memory than the available physical memory decrease significantly in performance due to the paging and swapping to get data from the slow disk to

the fast physical memory and vice versa.

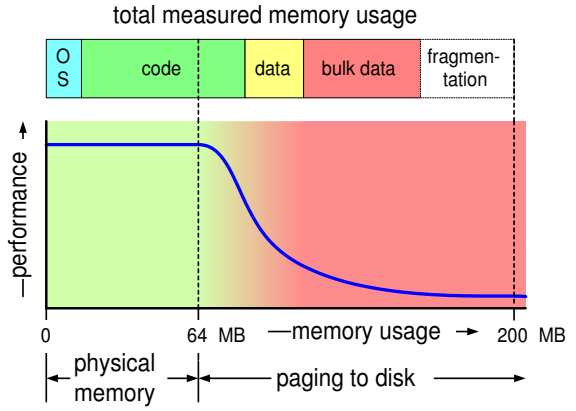


Figure 13.4: Memory usage half way R1

The analysis of additional measurements resulted in a decomposition of the memory used. The decomposition and the measurements are later used to allocate memory budgets. Figure 13.5 shows how the problem of poor performance was tackled, which is explained in much more detail in Chapter 15. The largest gains were obtained by the use of shared libraries, and by implementing an anti-fragmentation strategy for bulk data. Smaller gains were obtained by tuning, and analyzing the specific memory use more critical.

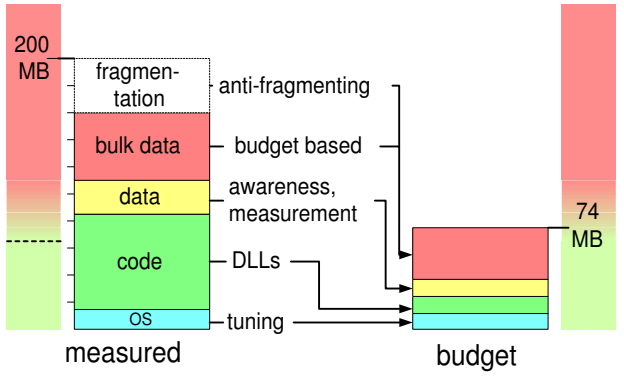


Figure 13.5: Solution of memory performance problem

Figure 13.6 shows the situation per process. Here the shared libraries are

shown separate of the processes. The category *other* is the accumulation of a number of small processes. This figure shows that every individual process did fit in the available amount of memory. A typical developer tests one process at a time. The developers did not experience a decreased performance caused by paging, because the system is not paging if only one process is active. At the time of integration, however, the processes are running on the same hardware concurrently and then the performance is suddenly very poor.

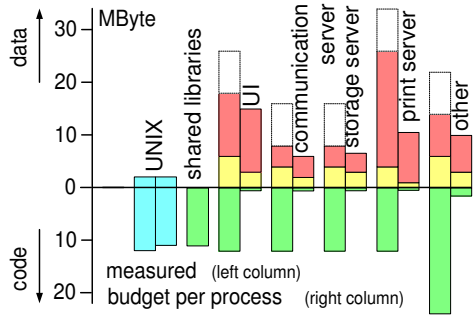


Figure 13.6: Visualization per process

Many other causes of performance problems have been found. All of these are shown in the annotated overlay on the software process structure in Figure 13.7.

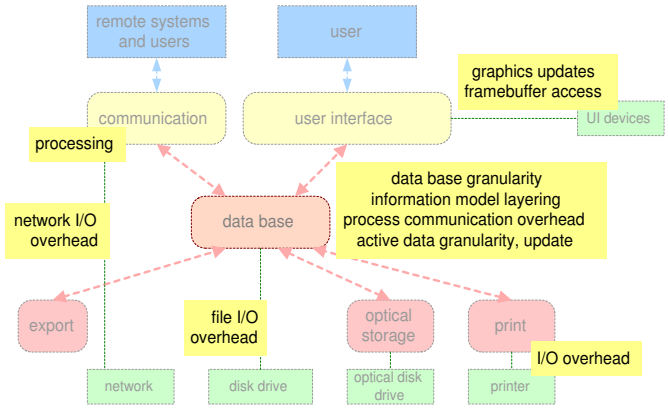


Figure 13.7: Causes of performance problems, other than memory use

Many of the performance problems are related to overhead, for instance for I/O and communication. A crucial set of design choices is related to granularity:

a fine grain design causes a lot of overhead. Another related design choice is the mechanism to be used: high level mechanisms introduce invisible overheads. How aware should an application programmer be of the underlying design choices?

For example, accessing patient information might result in an implicit transaction and query on the database. Building a patient selection screen by repeatedly calling such a function would cause tens to hundreds of transactions. With 25 ms per transaction this would result in seconds of overhead only to obtain the right information. The response becomes even worse if many layers of information have to be retrieved (patient, examination, study, series, image), resulting in even worse response time.

The rendering to the screen poses another set of challenges. The original *Basic Application* was built on Solaris 1, with the SunView windowing system. This system was very performance efficient. The product moved away from SunView, which was declared to be obsolete by the vendor, to the X-windowing system. The application and the windowing are running in separate processes. As a consequence all screen updates cause process communication overhead, including several copy operations of screen bitmaps. This problem was solved by implementing an integrated X-compatible screen manager running in the same process as the application, called Nix[2].

Interactive graphics require a fast response. The original brute force method to regenerate always the entire graphics object was too slow. The graphics implementation had to be redesigned, using damage area techniques to obtain the required responsiveness.

13.5 Safety

The clinical image quality can only be assessed by clinical stakeholders. Clinical stakeholders start to use the system, when the performance, functionality and reliability of the system is at a reasonable level. This reasonable level is achieved after a lot of integration effort has been spent. the consequence is that image quality problems tend to be detected very late in the integration. Most image quality problems are not recognized by the technology-oriented designers. The technical image quality (resolution, brightness, contrast) is usually not the problem.

Figure 13.8 shows a typical image quality problem that popped up during the integration phase. The pixel value x, corresponding to the amount of X-ray dose received in the detector, has to be transformed into a grey value $f(x)$ that is used

[2] A Dutch play on words: *niks* means nothing

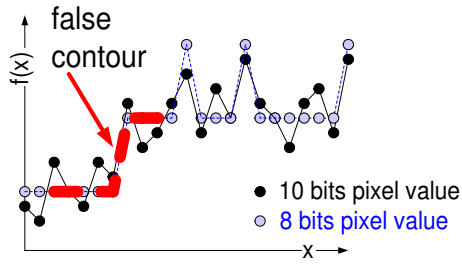


Figure 13.8: Image quality and safety problem: discretization of pixel values causes false contouring

to display the image on the screen. Due to discretization of the pixel values to 8 bits *false contours* become visible. For the human eye an artefact is visible between pixels that are mapped on a single grey value and neighboring pixels that are mapped on the next higher grey value. It is the levelling effect caused by the discretization that becomes visible as false contour. This artefact is invisible if the natural noise is still present. Concatenation of multiple processing steps can strongly increase this type of artifacts.

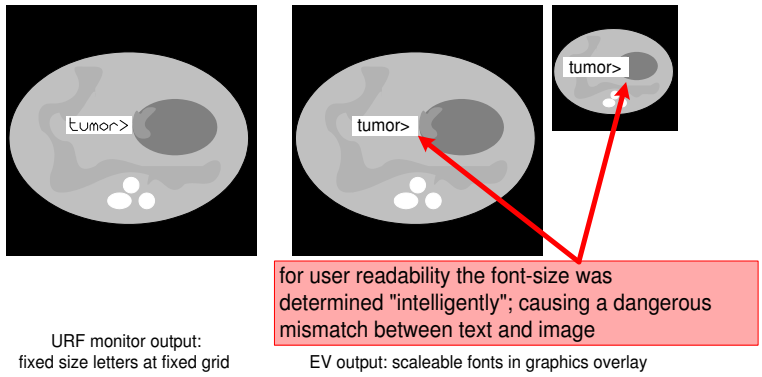


Figure 13.9: Safety problem caused by different text rendering mechanisms in the original system and in Easyvision

The original design of the viewing toolboxes provided scaling options for textual annotations, with the idea that the readability can be guaranteed for different viewport sizes. A viewport is a part of the screen, where an image and related information are shown. This implementation of the annotations on the X-ray system,

however, conflicts in a dangerous way with this model of scalable annotations, see Figure 13.9.

The annotations in the X-ray room are made on a fixed character grid. Sometimes the '>' and '<' characters are used as arrows, in the figure they point to the tumor. The text rendering in the medical imaging workstation is not based on a fixed character grid; often the texts will be rendered in variable-width characters. The combination of interface and variable-width characters is already quite difficult. The font scaling destroys the remaining part of the text-image relationship, with the immediate danger that the annotation is pointing to the wrong position.

The solution that has been chosen is to define an encompassing rectangle at the interface level and to render the text in a best fit effort within this encompassing rectangle. This strategy maintains the image-text relationship.

13.6 Summary

The development of the Easyvision RF started in 1991, with the perception that most of the software was available. During the developement phase it became clear that a significant amount of functionality had to be added in the area of printing. Chapter 14 will show the importance of the printing fumctionality. Performance and safety problems popped up during the integration phase. Chapter 15 will show the design to cope with these problems.

Chapter 14

Medical Imaging Workstation: CAF Views

14.1 Introduction

This chapter discusses the *Customer Objectives*, *Application* and *Functional* views of the Medical Imaging Workstation. Section 14.2 describes the radiology context. Section 14.3 describes the typical application of the system. Section 14.4 shows the key driver graph, from customer key drivers to system requirements, of the Medical Imaging Workstation. Section 14.5 shows the development of functionality of the family of medical imaging workstations in time. Section 14.6 discusses the need for standardization of information to enable interoperability of systems within the department and the broader scope of the hospital. The conclusion is formulated in section 14.7.

14.2 Radiology Context

The medical imaging workstation is used in the radiology department as an add-on to URF X-ray systems. The main objective of the radiologist is to provide diagnostic information, based on imaging, to the referring physician. In case of gastrointestinal problems X-ray images are used, where the contrast is increased by digestion of barium meal.

The work of the radiologist fits in an overall clinical flow, see Figure 14.1. The starting point is the patient visiting the family doctor. The family doctor can refer to a consultant; for gastrointestinal problems the consultant is an internist.

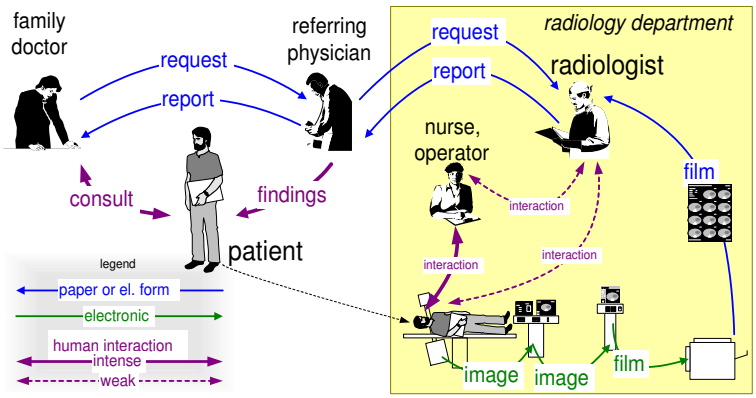


Figure 14.1: The clinical context of the radiology department, with its main stakeholders

The family doctor writes a request to this consultant. In the end the family doctor receives a report from the consultant.

Next the patient makes an appointment with the consultant. The consultant will do his own examination of the patient. Some of the examinations are not done by the consultant. Imaging, for example, is done by radiologist. From the viewpoint of the radiologist the consultant is the referring physician. The referring physician uses a request form to indicate the examination that is needed.

The patient makes an appointment via the administration of the radiology department. The administration will schedule the examination. The examination is done by hospital personnel (nurses, operator) under supervision of the radiologist. Most contact is between nurse and patient; contact between radiologist and patient is minimal.

The outcome of the imaging session in the examination room is a set of films with all the images that have been made. The radiologist will view these films later that day. He will dictate his findings, which are captured in written format and sent to the referring physician. The referring physician performs the overall diagnosis and discusses the diagnosis and, if applicable, the treatment with the patient.

The radiology department fits in a complex financial context, see Figure 14.2. The patient is the main subject from a clinical point of view, but plays a rather limited role in the financial flow. The patient is paying for insurance, which decouples him from the rest of the financial context.

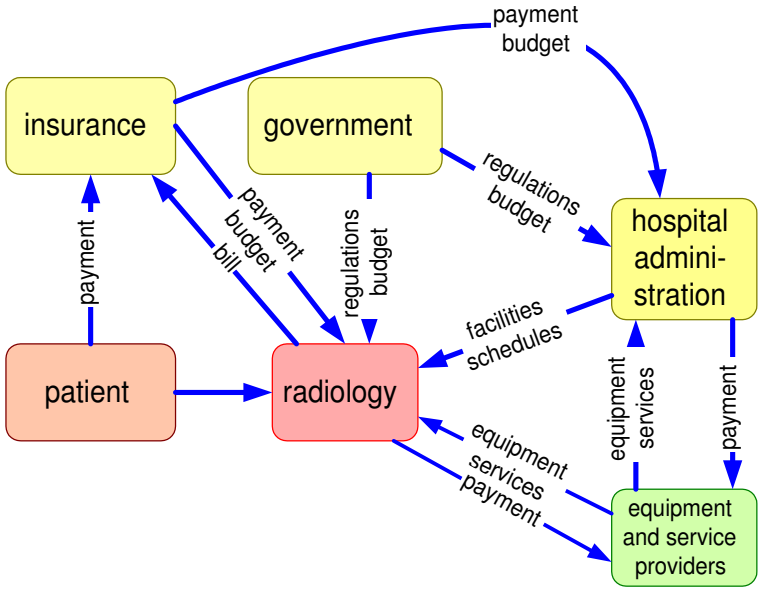


Figure 14.2: The financial context of the radiology department

The insurance company and the government have a strong interest in cost control[1]. They try to implement this by means of regulations and budgets. Note that these regulations vary widely over the different countries. France, for instance, has stimulated digitalization of X-ray imaging by higher reimbursements for digital images. The United States regulation is much less concerned with cost control, here the insurance companies participate actively in the health care chain to control the cost.

The hospital provides facilities and services for the radiology department. The financial decomposition between radiology department and hospital is not always entirely clear. They are mutually dependent.

The financial context is modeled in Figure 14.2 in a way that looks like the Calculating with Concepts technique, described by Dijkman et al in [22]. The diagram as it is used here, however, is much less rigorous as the approach of Dijkman. In this type of development the main purpose of these diagrams is building insight in the broader context. The rigorous understanding, as proposed by Dijkman, requires more time and is not needed for the purpose here. Most elements

[1] sometimes it even appears that that is the main interest, quality of health care appears than to be of secondary importance

in the diagram will not even have a formal interface with the product to be created. Note also that the diagram is a simplification of the reality: the exact roles and relations depend on the country, the culture and the type of department. For example a university hospital in France is different from a commercial imaging center in the USA. Whenever entities at this level are to be interfaced with the medical imaging workstation then an analysis is needed of the greatest common denominator to be able to define a rigorous interface.

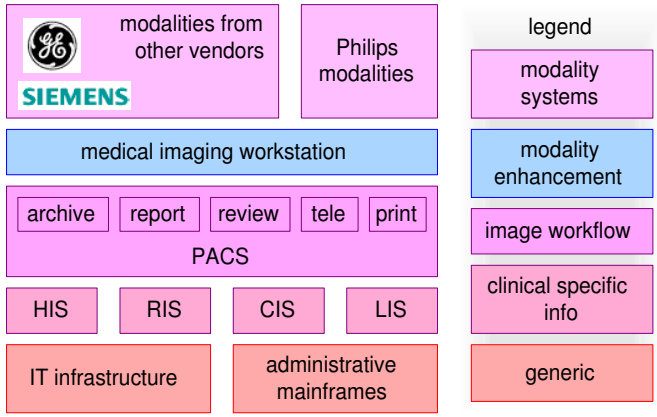


Figure 14.3: Application layering of IT systems

The medical imaging workstation is playing a role in the information flow in the hospital, it is part of the large collection of IT systems. Figure 14.3 shows a layered model of IT systems in the hospital, to position this product in the IT context. It is a layered model, where the lower layers provide the more generic functionality and the higher layers provide the more specific clinical imaging functionality.

In the hospital a normal generic IT infrastructure is present, consisting of networks, servers, PC's and mainframes. More specialized systems provide clinical information handling functions for different hospital departments (LIS for laboratory, CIS for cardio and RIS for radiology) and for the entire hospital (HIS Hospital Information System).

The generic imaging infrastructure is provided by the PACS (Picture Archiving and Communication System). This is a networked system, with more specialized nodes for specific functions, such as reporting, reviewing, demonstration, teaching and remote access.

The medical imaging workstation is positioned as a modality enhancer: an

add-on to the modality product to enhance productivity and quality of the examination equipment. The output of the modality enhancer is an improved set of viewable images for the PACS.

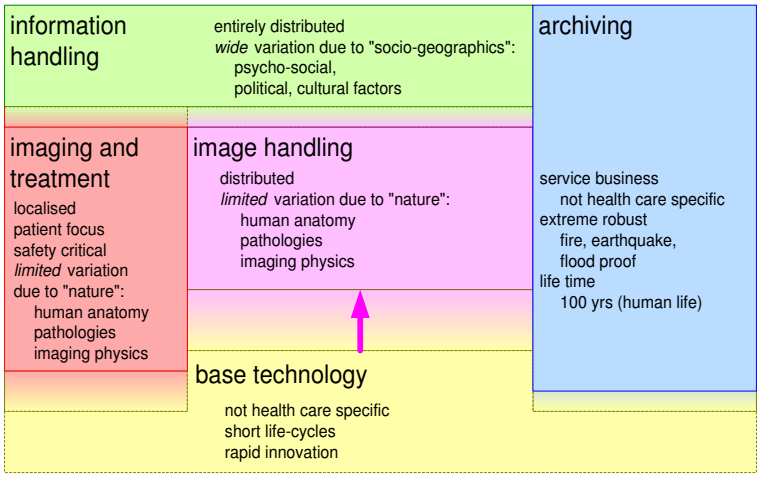


Figure 14.4: Reference model for health care automation

Figure 14.4 shows a reworked copy of the reference model for image handling functions from the “PACS Assessment Final Report”, September 1996 [18]. This reference model is classifying application areas on the basis of those characteristics that have a great impact on design decisions, such as the degree of distribution, the degree and the cause of variation and life-cycle.

Imaging and treatment functions are provided of modality systems with the focus on the patient. Safety plays an important role, in view of all kinds of hazards such as radiation, RF power, mechanical movements et cetera. The variation between systems is mostly determined by:

- the acquisition technology and its underlying physics principles.
- the anatomy to be imaged
- the pathology to be imaged

The complexity of these systems is mostly in the combination of many technologies at state-of-the-art level.

Image handling functions (where the medical imaging workstation belongs) are distributed over the hospital, with work-spots where needed. The safety related hazards are much more indirect (identification, left-right exchange). The variation

is more or less the same as the modality systems: acquisition physics, anatomy and pathology.

The *information handling* systems are entirely distributed, information needs to be accessible from everywhere. A wide variation in functionality is caused by "social-geographic" factors:

- psycho-social factors
- political factors
- cultural factors
- language factors

These factors influence what information must be stored (liability), or must not be stored (privacy), how information is to be presented and exchanged, who may access that information, et cetera.

The *archiving* of images and information in a robust and reliable way is a highly specialized activity. The storage of information in such a way that it survives fires, floods, and earthquakes is not trivial[2]. Specialized service providers offer this kind of storage, where the service is location-independent thanks to the high-bandwidth networks.

All of these application functions build on top of readily available IT components: the *base technology*. These IT components are innovated rapidly, resulting in short component life-cycles. Economic pressure from other domains stimulate the rapid innovation of these technologies. The amount of domain-specific technology that has to be developed is decreasing, and is replaced by base technology.

Figure 14.5 comes from the same report [18] showing the information flow within this reference model. During this flow the clinical value is increasing: annotations, comments, and anamnesis can be added during and right after the acquisition. The preparation for the diagnosis adds analysis results, optimizes layout and presentation settings, and pre-selects images. Finally the diagnosis is the required added value, to be delivered to the referring physician.

At the same time the richness of the image is decreasing. The richness of the image is how much can be done with the pixels in the image. The images after acquisition are very rich, all manipulation is still possible. When leaving the acquisition system the image is exported as a system independent image, where a certain trade-off between size, performance, image quality, and manipulation

[2]Today terrorist attacks need to be included in this list full of disasters, and secure needs to be added to the required qualities.

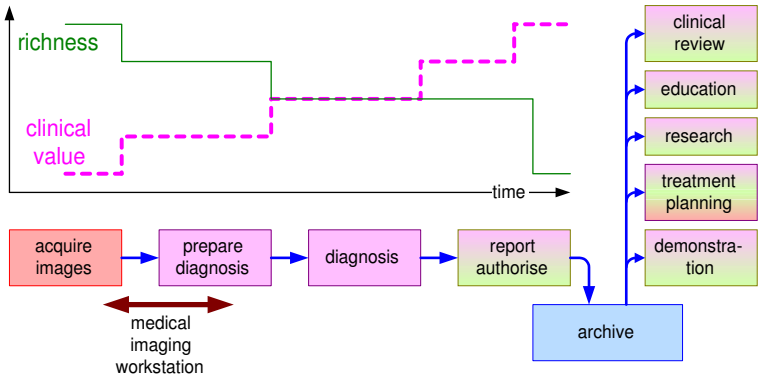


Figure 14.5: Clinical information flow

flexibility is made. This is an irreversible step in which some information is inherently lost. The results of the preparation for diagnosis are often frozen, so that no accidental changes can be made afterwards. Because this is the image used to diagnose, it is also archived to ensure liability. The archived result is similar to an electronic photo, only a limited set of manipulations can still be performed on it.

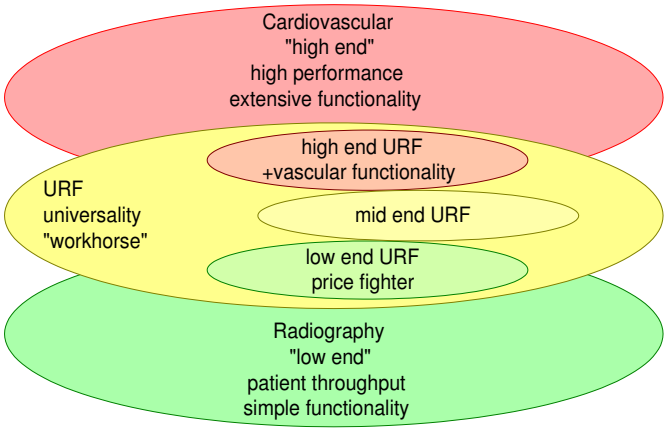


Figure 14.6: URF market segmentation

The first releases of the medical imaging workstation, as described in this case, are used in conjunction with URF (Universal Radiography Fluoroscopy) systems. This family of systems is a mid-end type of X-ray system, see Figure 14.6. At

the high end cardiovascular systems are used, with high clinical added value and a corresponding price tag. At the low end "radiography" systems offer straight forward imaging functionality, oriented at patient throughput. Approximately 70% of all X-ray examinations are radiographic exposures.

The URF systems overlap with cardiovascular and radiography market segments: high end URF systems also offer vascular functionality. Low end URF systems must fit in radiography constraints. The key driver of URF systems is the universality, providing logistic flexibility in the hospital.

14.3 Typical Case

The specification and design of the medical imaging workstation was based on "typical" cases. Figure 14.7 shows the typical case for URF examinations. Three examination rooms are sharing one medical imaging workstation. Every examination room has an average throughput of 4 patients per hour (patient examinations are interleaved, as explained below for Figure 14.8).

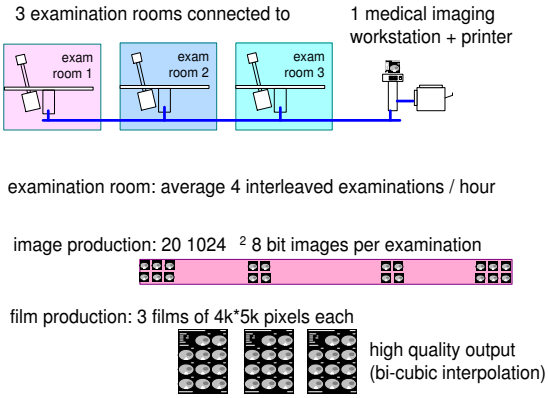


Figure 14.7: Typical case URF examination

The average image production per examination is 20 images, each of 1024^2 pixels of 8 bits. The images are printed on large film sheets with a size of approximately $24 * 30cm^2$. One film sheet consists of 4k by 5k pixels. The images must be sufficiently large to be easily viewed on the lightbox. These images are typically printed on 3 film sheets. Image quality of the film sheets is crucial, which translates into the use of bi-cubic interpolation.

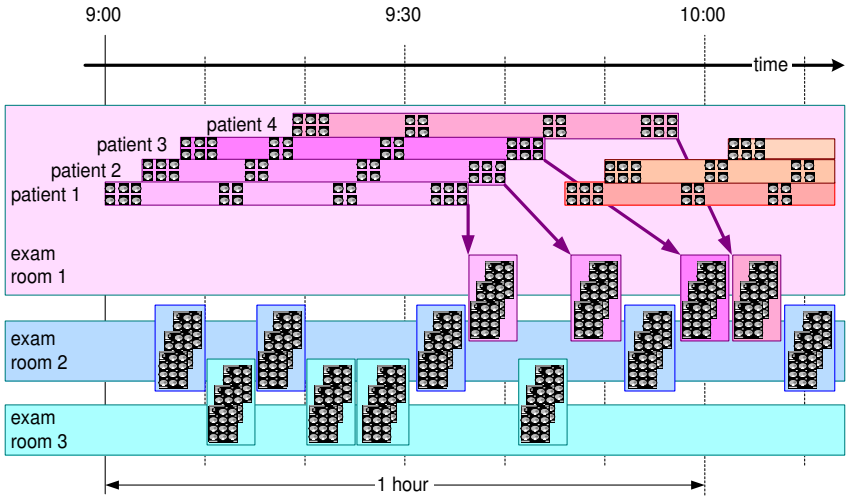


Figure 14.8: Timing of typical URF examination rooms

Figure 14.8 shows how patient examinations are interleaved. The patient is examined over a period of about one hour. This time is needed because the barium meal progresses through the intestines during this period. A few exposures are made during the passage of clinical relevant positions. The interleaving of patients in a single examination room optimizes the use of expensive resources. At the level of the medical imaging workstation the examinations of the different examination rooms are imported concurrently. The workstation must be capable of serving all three acquisition rooms with the specified typical load. The latency between the end of the examination and the availability of processed film sheets is not very critical.

14.4 Key Driver Graph

Figure 14.9 shows the key drivers from the radiologist point of view, with the derived application drivers and the related requirements, as described in Section 8.2. The graph is only visualized for the key drivers and the derived application drivers. The graph from application drivers to requirements is a many-to-many relationship, that becomes too complex to show in a single graph.

The key drivers are discussed in Subsections 14.4.1 to 14.4.5.

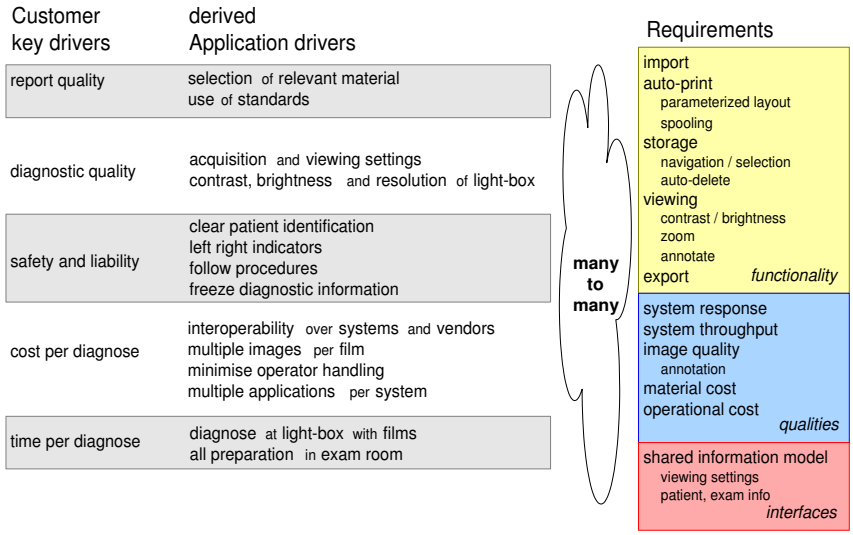


Figure 14.9: Key drivers, application drivers and requirements

14.4.1 Report Quality

The report quality determines the satisfaction of the referring physician, who is the customer of the radiologist. The layout, accessibility, and all these kind of factors determine the overall report quality. The radiologist achieves the report quality by:

selection of relevant material The selection of the material to be reported to the referring physician determines to a large degree the report quality.

use of standards The use of standard conventions, for instance pathology classification, improves the report quality.

14.4.2 Diagnostic Quality

The diagnostic quality is the core of the radiologist's work. The diagnostic quality is achieved by:

acquisition and viewing settings The actual acquisition settings and the related viewing settings have a great impact on the visibility of the pathology and anatomy.

contrast, brightness and resolution of lightbox The lightbox has a very good diagnostic image quality: high brightness, high resolution, and many images can be shown simultaneously.

14.4.3 Safety and Liability

Erroneous diagnoses are dangerous for the patient; the radiologist might be sued for mistakes. Also mistakes in the related annotations (wrong patient name, wrong position) are a safety risk for the patient and hence a liability risk for the radiologist. The derived application drivers for safety and liability are:

clear patient identification Erroneous patient identification is a safety risk.

left right indicators Erroneous positioning information is a safety risk. Left-right exchanges are notoriously dangerous.

follow procedures Clinical procedures reduce the chance of human errors. Following these procedures lowers the liability for the radiologist.

freeze diagnostic information Changing image information after the diagnosis is a liability risk: different interpretations are possible, based on the changes.

14.4.4 Cost per Diagnosis

Insurance and government generate a lot of cost pressure. Cost efficiency can be expressed in cost per diagnosis. The cost per diagnosis is reduced in the following ways:

interoperability over systems and vendors Mix and match of systems, not constrained by vendor or system lock-ins, allow the radiology department to optimize the mix of acquisition systems to the local needs.

multiple images per film Film is a costly resource (based on silver). Efficiency of film real estate is immediately cost efficient. A positive side effect is that film efficiency is also beneficial for viewing on the lightbox, because the images are then put closer together.

minimize operator handling Automation of repeated actions will reduce the amount of personnel needed, which again is a cost reduction. An example is the use of predefined and propagated settings that streamline the flow of information. This is a cost reduction, but most of all it improves the convenience for the users.

multiple applications per system Universality of acquisition system and workstation provides logistics flexibility in the radiology department. This will in the end result in lower cost.

14.4.5 Time per Diagnosis

Time efficiency is partially a cost factor, see 14.4.4, but it is also a personal satisfaction issue for the radiologist. The time per diagnosis is reduced by the following means:

diagnose at lightbox with films This allows a very fast interaction: zooming is done by a single head movement, and the next patient is reached by one button, that exchanges the films mechanically in a single move.

all preparation in exam room The personnel operating the examination room also does the preparation for the diagnosis. This work is done on the fly, interleaved with the examination work.

14.4.6 Functional Requirements

The functionality that is needed for to realize the derived application drivers is:

import The capability to import data into the workstation data store in a meaningful way.

autoprint The capability to print the image set without operator intervention:

- **parametrized layout** Film layout under control of the remote acquisition system.
- **spooling** Support for concurrent import streams, which have to be printed by a single printer.

storage The capability to store about one day of examinations at the workstation, both as a buffer and to enable later review:

- **navigation/selection** The capability to find and select the patient, examination and images.
- **autodelete** The capability to delete images when they are printed and no longer needed. This function allows the workstation to be used in an operator free server. The import, print and auto-delete run continuously as a standard sequence.

viewing All functions to show and manipulate images, the most frequently used subset:

contrast/brightness Very commonly used grey-level user interface.

zoom Enlarge part of the image.

annotate Add textual or graphic annotations to the image.

export Transfer of images to other systems.

Note that the *import*, *storage* and *autoprint* functionality are core to satisfy the key drivers, while the viewing and export functionality is only *nice to have*.

14.4.7 Quality Requirements

The following qualities need to be specified quantitatively:

system response Determines the speed and satisfaction of preparing the diagnosis by means of the workstation.

system throughput As defined by the typical case.

image quality Required for preparation of the diagnosis on screen and for diagnosis from film. Specific quality requirements exists for the relation between image and annotation:

annotation The relation between annotation and image is clinically relevant and must be reproducible.

material cost The cost price of the system must fit in the cost target.

operational cost The operational cost (cost of consumables, energy, et cetera) must fit in the operational target.

14.4.8 Interface Requirements

Key part of the external interfaces is the shared information model that facilitates interoperability between different systems. The cooperating systems must adhere to a shared information model. Elements of such an information model are:

viewing settings Sharing the same presentation model to guarantee the same displayed image at both systems.

patient, exam info Sharing the same meta information for navigation and identification.

14.5 Functionality

Figure 14.10 shows a retrospective overview of the development of functionality over time. The case described here focuses on the period 1992, and 1993. However the vision of the product group was to design a platform that could serve many applications and modalities. The relevance of this retrospective overview is to show the expected (and realized!) increase of functionality.

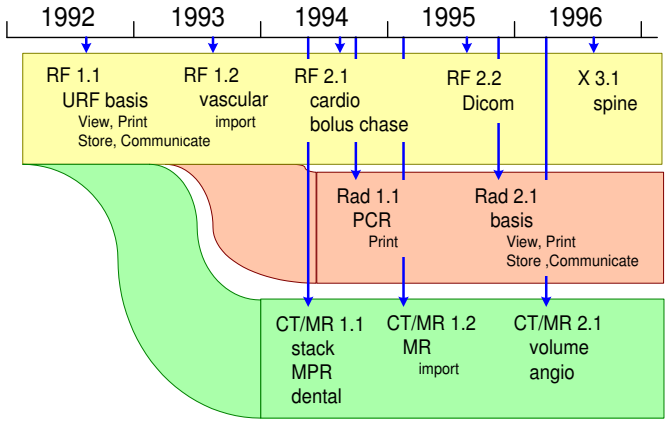


Figure 14.10: Retrospective functionality roadmap

The first release of the product served the URF market and provided the so-called view-print-store-communicate functionality. We already saw in figure 14.9 that a lot of functionality is hidden in this simple quartet.

Release 1.2 added import from vascular systems to the functionality. Cardio import and functionality and bolus chase reconstruction were added in release 2.1. Cardio functionality in this release consisted mostly of analysis functions, such as cardiac volume and wall motion analysis. The bolus chase reconstruction takes a series of exposures as input an fuses them together into a single large overview, typically used to follow the bolus chase through the legs.

Release 2.2 introduced DICOM as the next generation of information model standard. The first releases were based on the ACR/NEMA standard, DICOM succeeded this standard. Note that the installed base required prolongation of ACR/NEMA-based image exchange. Release 3.1 added spine reconstruction and analysis. The spine reconstruction is analogous to the bolus chase reconstruction, however spine specific analysis was also added.

On the basis of the URF-oriented R1.1 workstation a CT/MR workstation was

developed, which was released in 1994. CT/MR images are slice-based (instead of projection-based as in URF), which prompted the development of a stack view application (fast scrolling through a stack of images). Reconstruction of oblique and curved slices is supported by means of MPR (Multi Planar Reformatting). A highly specialized application was built on top of these applications. This was a dental package, allowing viewing of the jaws, with the molars, and with the required cross sections.

Release 2.1 of the CT/MR workstation added a much more powerful volume viewing application and a more specialized angio package, with viewing and analysis capability.

Also derived from the RF workstation a radiography workstation was built. R1.1 of this system was mostly a print server, while R2.1 supported the full view-print-store-communicate functionality.

The *commercial*, *service* and *goods flow* decompositions were present as part of the formalized documentation (TPD).

14.6 Interoperability via Information Model

The health care industry is striving for interoperability by working on standard exchange formats and protocols. The driving force behind this standardization is the ACR/NEMA, in which equipment manufacturers participate in the standardization process.

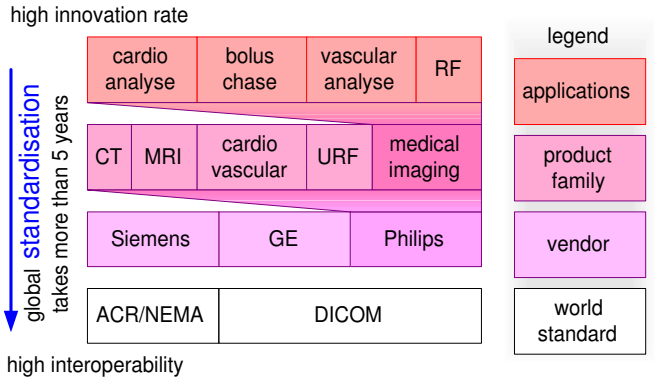


Figure 14.11: Information model, standardization for interoperability

Standardization and innovation are often opposing forces. The solution is

often found in defining an extendable format. and in standardization of the mature functionality. Figure 14.11 shows the approach as followed by the medical imaging product group. The communication infrastructure and the mature application information is standardized in DICOM. The new autoprint functionality was standardized at vendor level. Further standardization of autoprint is pushed via participation in DICOM work groups.

A good strategy is to use the standard data formats as much as possible, and to build vendor specific extensions as long as the required functionality is not yet standardized. The tension between standardization and innovation is also present at many levels: between vendors, but also between product groups in the same company and also between applications within the same product. At all levels the same strategy is deployed. Product family specific extensions are made as long as no standard vendor solution is available.

This strategy serves both needs: interoperability for mature, well defined functionality and room for innovative exploration.

The information model used for import, export and storage on removable media is one of the most important interfaces of these systems. The functionality and the behavior of the system depend completely on the availability and correctness of this information. The specification of the information model and the level of adherence and the deviations is a significant part of the specification and the specification effort. A full time architect created and maintained this part of the specification.

14.7 Conclusion

The context of the system in the radiology department has been shown by means of multiple models and diagrams: clinical context with stakeholders, financial context, application layers in IT systems, a reference model for health care automation, clinical information flow, and URF market segmentation. Figure 14.12 shows the coverage in actual documentation of the submethods discussed in part II. The actual documentation of the *Customer Objectives* and *Application* views was quite poor, as indicated in Figure 14.12. Most of the models and diagrams shown here were not present in the documentation of 1992. The application of the system has been shown as typical case. The typical case was documented explicitly in 1992. The key driver graph, discussed in Section 14.4, is also a reconstruction in retrospect. The limited attention for the *Customer Objectives* and *Application* views is one of the main causes of the late introduction of printing

functionality.

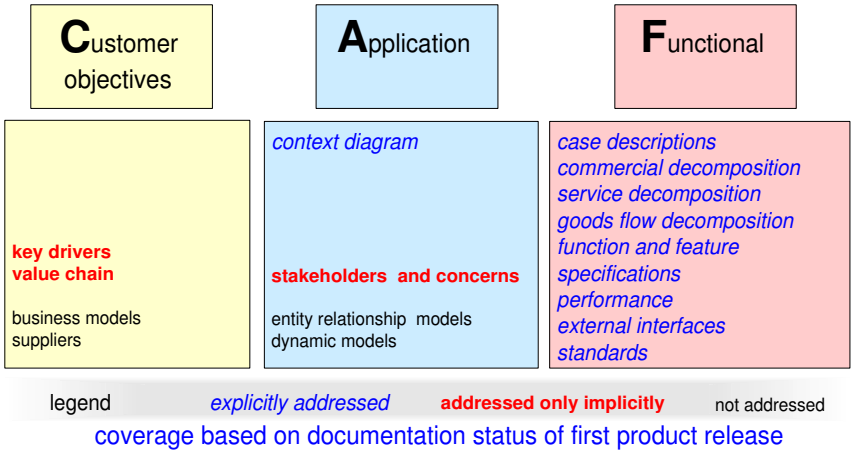


Figure 14.12: Coverage of submethods of the CAF views

The functional view was well documented in 1992. The functions and features have been discussed briefly in Section 14.5. The functions and features were well documented in so-called *Functional Requirement Specifications*. Interoperability, discussed briefly in Section 14.6, was also documented extensively. Figure 14.12 shows that the coverage of the *Functional* view is high.

Chapter 15

Medical Imaging Workstation: CR Views

15.1 Introduction

The conceptual and realization views are described together in this chapter. The realization view, with its specific values, brings the concepts more alive.

Section 15.2 describes the processing pipeline for presentation and rendering, and maps the user interface on these concepts. Section 15.4 describes the concepts needed for memory management, and zooms in on how the memory management is used to implement the processing pipeline. Section 15.3 describes the software architecture. Section 15.5 describes how the limited amount of CPU power is managed.

The case material is based on actual data, from a complex context with large commercial interests. The material is simplified to increase the accessibility, while at the same time small changes have been made to remove commercial sensitivity. Commercial sensitivity is further reduced by using relatively old data (between 8 and 13 years in the past). Care has been taken that the value of the case description is maintained.

15.2 Image Quality and Presentation Pipeline

The user views the image during the examination at the console of the X-ray system, mostly to verify the image quality and to guide the further examination. Later the same image is viewed again from film to determine the diagnosis and to

prepare the report. Sometimes the image is viewed before making a hardcopy to optimize the image settings (contrast, brightness, zoom). The user expects to see the same image at all work-spots, independent of the actual system involved.

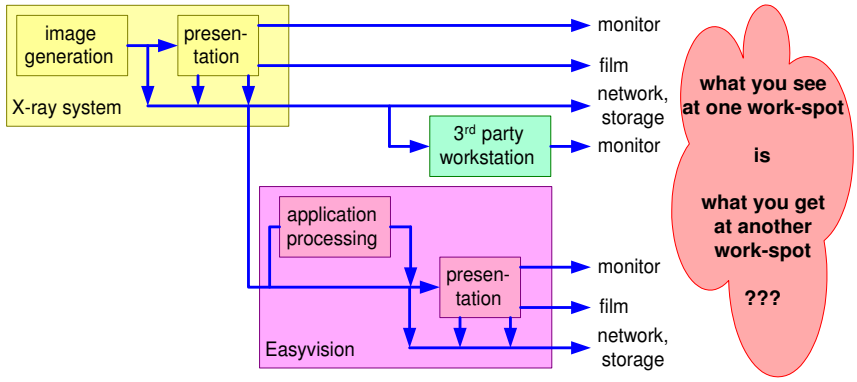


Figure 15.1: The user expectation is that an image at one work-spot looks the same as at other work-spots. This is far from trivial, due to all data paths and the many parties that can be involved

Figure 15.1 shows many different possible work-spots, with different media. The user expects *What You See Is What You Get* (WYSIWYG) everywhere. From an implementation point of view this is far from trivial. To allow optimal handling of images at other locations most systems export images halfway their internal processing pipeline: acquisition specific processing is applied, rendering specific processing is not applied, but the rendering settings are transferred instead. All systems using these intermediate images need to implement the same rendering in order to get the same image perception. The design of these systems is strongly coupled, due to the shared rendering know-how.

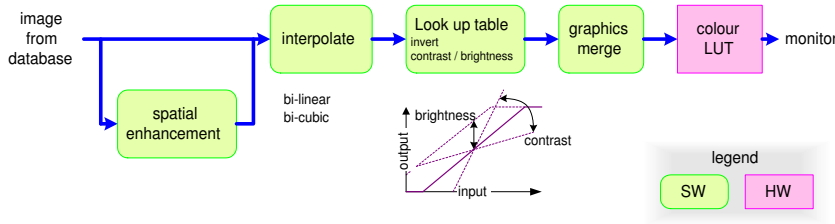


Figure 15.2: The standard presentation pipeline for X-ray images

Figure 15.2 shows the rendering pipeline as used in the medical imaging workstation. Enhancement is a filter operation. The coefficients of the enhancement kernel are predefined in the acquisition system. The interpolation is used to resize the image from acquisition resolution to the desired view-port (or film-port) size. The grey-levels for display are determined by means of a lookup table. A lookup table (LUT) is a fast and flexible implementation of a mapping function. Normally the mapping is linear: the slope determines the contrast and the vertical offset the brightness of the image. Finally graphics and text are superimposed on the image, for instance for image identification and for annotations by the user.

The image interpolation algorithm used depends on desired image quality and on available processing time. Bi-linear interpolation is an interpolation with a low-pass filter side effect, by which the image becomes less sharp. An ideal interpolation is based on a convolution with a sinc-function ($sin(x)/x$). A bi-cubic interpolation is an approximation of the ideal interpolation. The bi-cubic interpolation is parameterized. The parameter settings determine how much the interpolation causes low pass or high pass filtering (blurring or sharpening). These bi-cubic parameter choices are normally not exported to the user interface, the selection of values requires too much expertise. Instead, the system uses empirical values dependent on the interpolation objective.

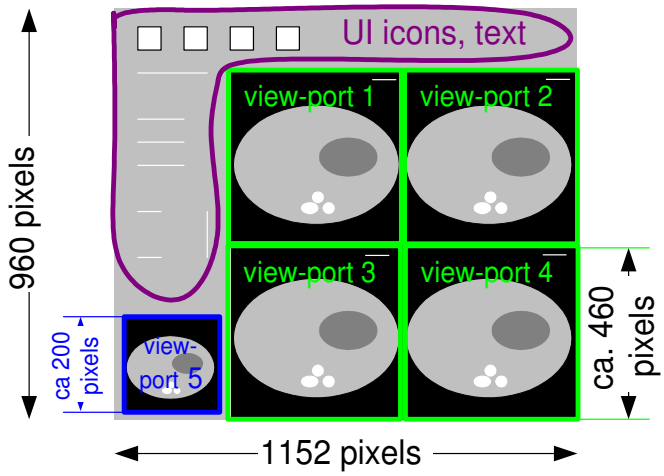


Figure 15.3: Quadruple view-port screen layout

The monitor screen is a scarce resource of the system, used for user interface control and for the display of images. The screen is divided in smaller rectangular

windows. Windows displaying images are called view-ports. Every view-port uses its own instantiation of a viewing pipeline. Figure 15.3 shows an example of a screen layout, viewing four images simultaneously. At the bottom left a fifth view-port is used for navigational support, for instance in case of zooming this view-port functions as a roadmap, enabling direct manipulation of the zoom-area. The fifth view-port also has its own viewing pipeline instance.

The concepts visible in this screen layout are view-ports, icons, text, an image area (with the 4 main view-ports), and a user interface area with navigation support. The figure adds a number of realization facts, such as the total screen-size, and the size of the view-ports. The next generation of this system used the same concepts, but the screen size was 1280*1024, resulting in slightly larger view-ports and a slightly larger ratio between image area and user interface area.

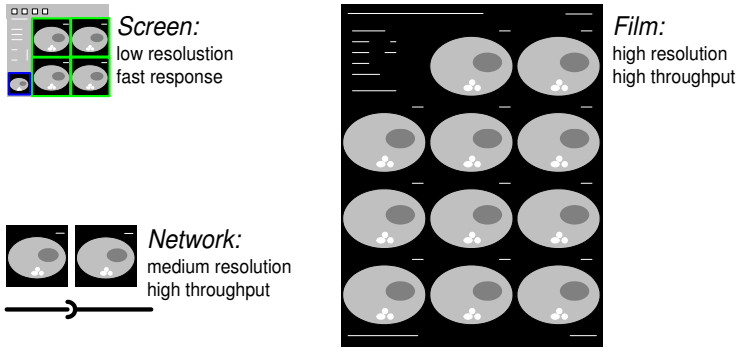


Figure 15.4: Rendered images at different destinations

At all places where source images have to be rendered into viewable images an instance of the presentation pipeline is required. Note that the characteristics of the usage of the presentation pipeline in these different processes vary widely. Figure 15.4 shows three different destinations for rendered images, with the different usage characteristics.

15.3 Software Specific Views

The execution architecture of Easyvision is based on UNIX-type processes and shared libraries. Figure 15.5 shows the process structure of Easyvision. Most processes can be associated with a specific hardware resource, as shown in this figure. Core of the Easyvision software architecture is the database. The database pro-

vides *fast, reliable, persistent* storage and it provides *synchronization* by means of active data. The concept of active data is based on the *publish-subscribe pattern* [25] that allows all users of a some information to be notified when changes in the information occur. Synchronization and communication between processes always takes place via this database.

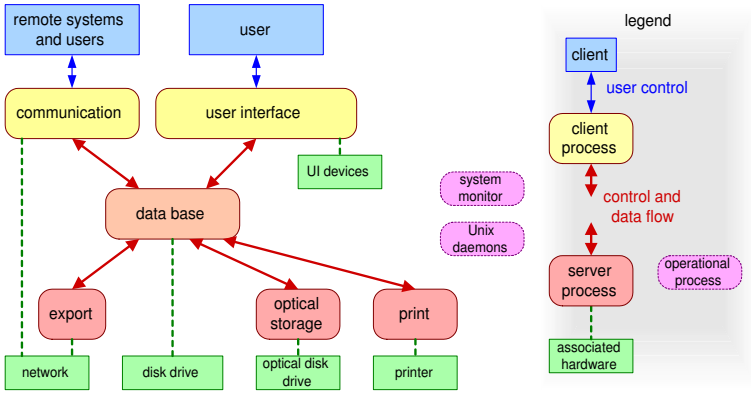


Figure 15.5: Software processes or tasks running concurrently in Easyvision

Figure 15.5 shows four types of processes: *client processes*, *server processes database process*, and *operational processes*. A client interacts with a user (remote or direct), while the servers perform their work in the background. The database connects these two types of processes. Operational processes belong to the computing infrastructure. Most operational processes are created by the operating system, the so called daemons. The system monitoring processes is added for exception handling purposes. The system monitor detects hanging processes and takes appropriate action to restore system operation.

A process as unit of design is used for multiple reasons. The criteria used to determine the process decomposition are:

management of concurrency Activities that are concurrent run in separate processes.

management of shared devices A shared device is managed by a server process.

unit of memory budget Measurement of memory use at process level is supported by multiple tools.

unit of distribution over multiple processors A process can be allocated to a processor, without the need to change the code within the process.

unit of exception handling Faults are contained within the process boundaries. The system monitor observes at process level, because the operating system provides the means at process level.

Manageability, visibility and understandability benefit from a limited number of processes. One general rule is to minimize the amount of processes, in the order of magnitude of ten processes.

The presentation pipeline, as depicted in Figure 15.2, is used in the *user interface* process, the *print* server and the *export server*.

Figure 15.6 shows the software from the dependency point of view. Software in higher layers depends on, has explicit knowledge of, lower layers of the software. Software in the lower layers should not depend on, or have explicit knowledge of software in higher layers.

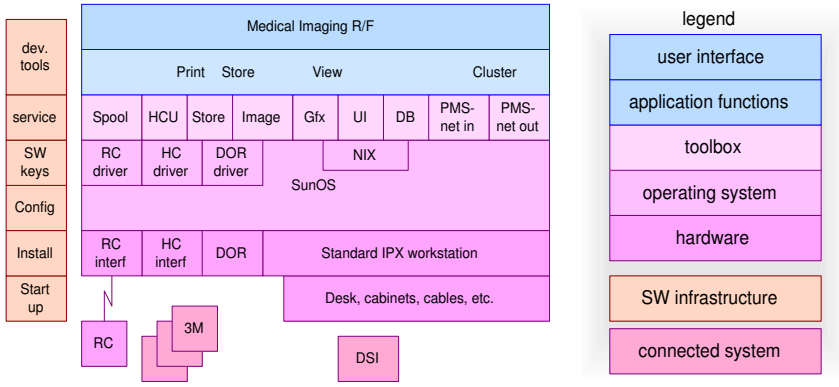


Figure 15.6: Simplified layering of the software

The caption of Figure 15.6 explicitly states this diagram to be simplified. The original design of this software did not use the layering concept. The software has been restructured in later years to make the dependency as layering explicit. The actual number of layers based on larger packages did exceed 15. Reality is much more complex than this simplified diagram suggests.

15.4 Memory Management

The amount of memory in the medical imaging workstation is limited for cost reasons, but also for simple physical reasons: the workstation used at that moment did not support more than 64 MByte of physical memory. The workstation and

operating system did support virtual memory, but for performance reasons this should be used sparingly.

	code		object data		bulk data		total	
memory budget in Mbytes	R1	R2	R1	R2	R1	R2	R1	R2
shared code	6.0	11.0					6.0	11.0
UI process	0.2	0.3	2.0	3.0	12.0	12.0	14.2	15.3
database server	0.2	0.3	4.2	3.2		3.0	4.4	6.5
print server	0.4	0.3	2.2	1.2	7.0	9.0	9.6	10.5
DOR server	0.4	0.3	4.2	2.0	2.0	1.0	6.6	3.3
communication server	1.2	0.3	15.4	2.0	10.0	4.0	26.6	6.3
UNIX commands	0.2	0.3	0.5	0.2			0.7	0.5
compute server		0.3		0.5		6.0		6.8
system monitor		0.3		0.5				0.8
application total	8.6	13.4	28.5	12.6	31.0	35.0	66.1	61.0
UNIX							7.0	10.0
file cache							3.0	3.0
total							76.1	74.0

Figure 15.7: Memory budget of Easyvision release 1 and release 2

A memory budget is used to manage the amount of memory in use. Figure 15.7 shows the memory budgets of release 1 and release 2 of Easyvision RF side by side. Three types of memory are distinguished: *program or code*, read-only from operating system point of view, *object data*, dynamically allocated and deallocated in a heap-based fashion, and *bulk data* for large consecutive memory areas, mostly used for images.

Per process, see Section 15.3, the typical amount of memory per category is specified. The memory usage of the operating system is also specified. The dynamic libraries, that contain the code shared between processes, is explicitly visible in the budget.

The figure shows the realization for two successive releases, for which we can observe that the concepts are stable, but that the realization changes significantly. Release 1 used a rather straightforward communication server, operating on all import streams in parallel, keeping everything in memory. This is very costly with respect to memory. R2 serializes the memory use of different import streams and uses the memory in a more pipelined way. These changes result in a significant reduction of the memory being used. In the same time frame the supplier dictated a new operating system, SunOS was end-of-life and was replaced by Solaris 2. This had a negative impact on the memory consumption; the budget shows an increase of 7 MByte to 10 MByte for the UNIX operating system.

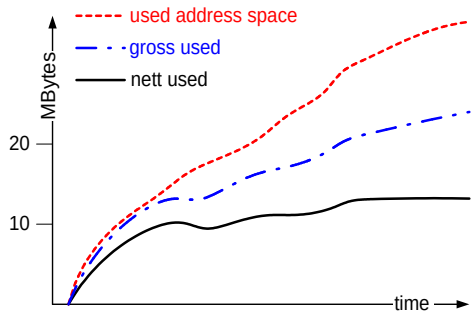


Figure 15.8: Memory fragmentation increase. The difference between gross used and nett used is the amount of unusable memory due to fragmentation

The decomposition in object data and bulk data is needed to prevent memory fragmentation. Fragmentation of memory occurs when the allocation is dynamic with different sizes of allocated memory over time. The fragmentation increases over time. Due to the paging of the virtual memory system not all fragmentation is disastrous. Figure 15.8 shows the increase of the amount of memory over time. The net amount of memory stabilizes after some time, but the gross amount of memory increases due to ongoing fragmentation. The amount of virtual memory in use (and the address space) is increasing even more, however a large part of this virtual memory is paged out and is not really a problem.

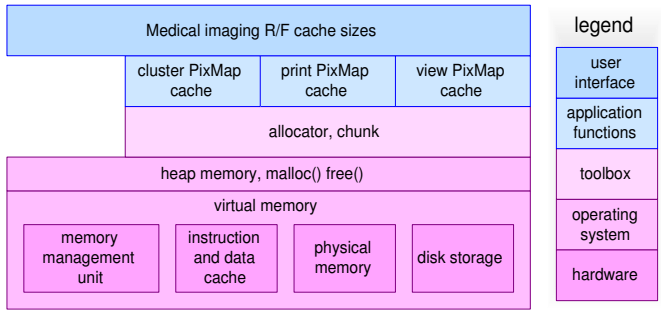


Figure 15.9: Cache layers at the corresponding levels of Figure 15.6

The hardware and operating system support fast and efficient memory-based on hardware caching and virtual memory, the lowest layer in Figure 15.9. The application allocates memory via the heap memory management functions mal-

loc() and free(). From an application point of view a sheer infinite memory is present, however the speed of use depends strongly on the access patterns. Data access with a high locality are served by the data cache, which is the fastest (and smallest) memory layer. The next step in speed and size (slower, but significantly larger) is the physical memory. The virtual memory, mostly residing on disk, is the slowest but largest memory layer.

The application software does not see or control the hardware cache or virtual memory system. The only explicit knowledge in the higher software layers of these memory layers is in the dimensioning of the memory budgets as described later.

The toolbox layer provides anti-fragmentation memory management. This memory is used in a cache like way by the application functions, based on a *Least Recently Used* algorithm. The size of the caches is parameterized and set in the highest application layer of the software.

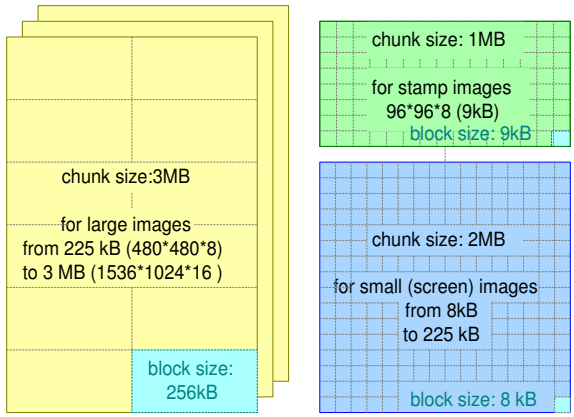


Figure 15.10: Memory allocators as used for bulk data memory management in Easyvision RF

The medical imaging workstation deploys pools with fixed size blocks to minimize fragmentation. A two level approach is taken: pools are allocated in large *chunks*, every chunk is managed with fixed size *blocks*. For every chunk is defined which bulk data sizes may be stored in it.

Figure 15.10 shows the three chunk sizes that are used in the memory management concepts chunks, block sizes and bulk data sizes as used in Easyvision RF. One chunk of 1 MByte is dedicated for so-called *stamp* images, 96*96 down scaled images, used primarily for visual navigation (for instance pictorial index).

The block size of 9 kbytes is exactly the size of a stamp image. A second chunk of 3 MBytes is used for large images, for instance images with the original acquisition resolution. Small images, such as images at display resolution, will be allocated in the third chunk of 2 MBytes. The dimensioning of the block and chunk sizes is based on a priori know-how of the application of the system, as described in Section 14.3. The block sizes in the latter two chunks are 256 kbytes for large images and 8 kbytes for small images. These block sizes result in balanced and predictable memory utilization and fragmentation within a chunk.

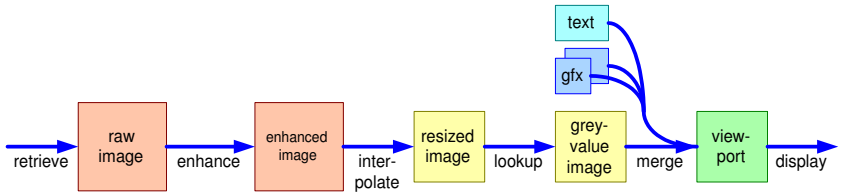


Figure 15.11: Intermediate processing results are cached in an application level cache

The chunks are used with cache like behavior: images are kept until the memory is needed for other images. Figure 15.11 shows the cached intermediate results. This figure is a direct transformation of the viewing pipeline in Figure 15.2, with the processing steps replaced by arrows and the data-arrows replaced by stores. In Section 15.5 the gain in response time is shown, which is obtained by caching the intermediate images.

Figure 15.12 shows how the *chunks* are being used in quadruple viewing (Figure 15.3). The 1024^2 images with a depth of 1 or 2 bytes will be stored in the 3 MB chunks. The smaller interpolated images of 460^2 will go into the 2 MB chunks, requiring 27 blocks of 8kB for an 1 byte pixel depth or 54 blocks for 2 2 bytes per pixel. Also the screen size images of the navigation view-port fall in the range that maps on the 2 MB chunk, requiring 5 blocks per 200^2 image.

Everything added together requires more blocks than available in the 2 and 3 MB chunks. The cache mechanism will sacrifice the least recently used intermediate results.

For memory and performance reasons the navigation view-port is using the stamp image as source image. This image, which is shown in a small view-port at the left hand side of the screen, is only used for navigational support of the user interface. Response time is here more important than image quality.

The print server uses a different memory strategy than the user interface pro-

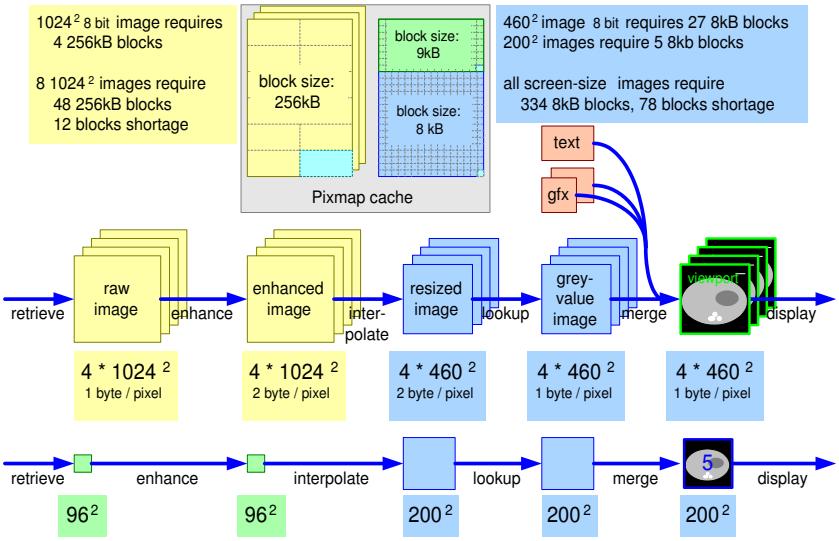


Figure 15.12: Example of allocator and cache use. In this use case not all intermediate images fit in the cache, due to a small shortage of blocks. The performance of some image manipulations will be decreased, because the intermediate images will be regenerated when needed.

cess, see Figure 15.13. The print server creates the film-image by rendering the individual images. The film size of 4k*5k images is too large to render the entire film at once in memory: 20 Mpixels, while the memory budget allows 9 Mbyte of bulk data usage. The film image itself is already more than the provided memory budget!

The film image is built up in horizontal bands, which are sent to the laser printer. The size of the stroke is chosen such that input image + intermediate results + 2 bands (for double buffering) fit in the available bulk data budget. At the same time the band should not be very small because the banding increases the overhead and some duplicate processing is sometimes needed because of edge effects.

The print server uses the same memory management concepts as shown in the figure with cache layers, Figure 15.9. However the application level caching does not provide any significant value for this server usage, because the image data flow is straightforward and predictable.

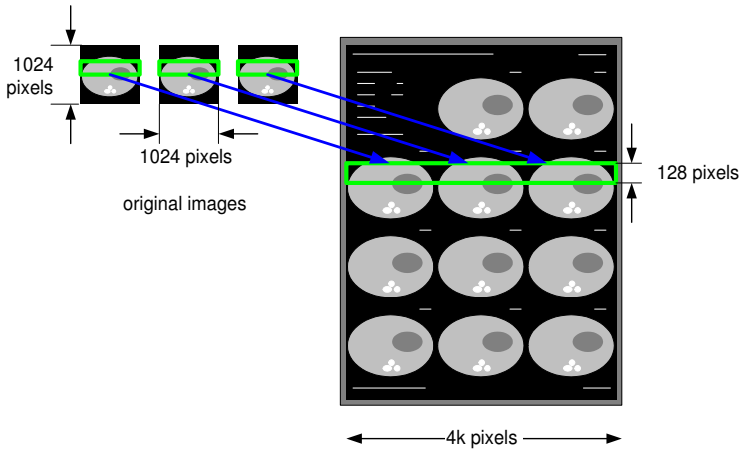


Figure 15.13: Print server is based on different memory strategy, using bands

15.5 CPU Usage

The CPU is a limited resource for the Easyvision. The performance and throughput of the system depend strongly on the available processing power and the efficiency of using the processing power. CPU time and memory can be exchanged partially, for instance by using caches to store intermediate results.

Figure 15.14 shows typical update speeds and processing times for a single image user interface layout. Contrast brightness (C/B in the figure) changes must be fast, to give immediate visual feedback when turning a contrast or brightness wheel. Working on the cached resized image about 7 updates per second are possible, which is barely sufficient. The gain of the cached design relative to the non-cached design is about a factor 8 (7 updates per second versus 0.9 updates per second). Zooming and panning is done with an update rate of 3 updates per second. The performance gain for zooming and panning is from application viewpoint less important, because these functions are used only exceptionally in the daily use.

Retrieving the next image (also a very frequent user operation), requires somewhat more than a second, which was acceptable at that moment in time. This performance is obtained by slightly compromising the image quality: a bilinear interpolation is used for resizing, instead of the better bi-cubic interpolation. For the monitor, with its limited resolution this is acceptable, for film (high resolution, high brightness) bi-cubic interpolation is required.

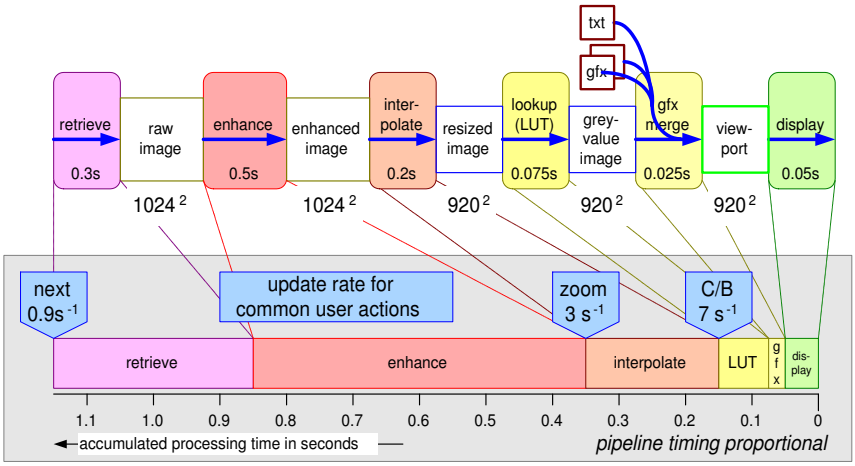


Figure 15.14: The CPU processing times are shown per step in the processing pipeline. The processing times are mapped on a proportional time line to visualize the viewing responsiveness

For background tasks a CPU budget is used, expressed in CPU seconds per Mega-byte or Mega-pixel. This budget is function-based: importing and printing. Most background jobs involve a single server plus interaction with the database server.

Two use cases are relevant: interactive viewing, with background jobs, and pure print serving. For interactive response circa 70% of CPU time should be available, while the load of printing for three examination rooms, which is a full throughput case, must stay below 90% of the available CPU time. Figure 15.15 shows the load for serving a single examination room and for serving three examination rooms. Serving a single examination room takes 260 seconds of CPU time per examination of 15 minutes, leaving about 70% CPU time for interactive viewing. Serving three examination rooms takes 13 minutes of CPU time per 15 minutes of examinations, this is just below the 90%.

15.6 Measurement Tools

The resource design as described above is supported in the implementation by means of a few simple, but highly effective measurement tools. The most important tools are: *Object Instantiation Tracing*, *standard Unix utilities* and a *heap*

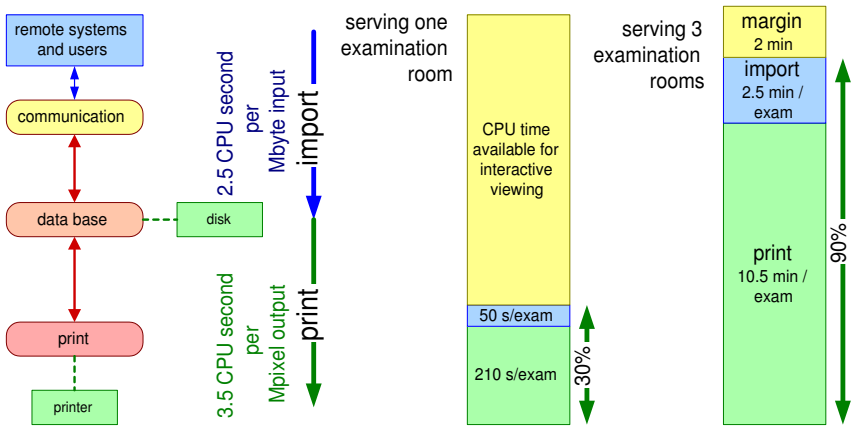


Figure 15.15: Server CPU load. For a single examination room sufficient CPU time is left for interactive viewing. Serving three examination rooms fits in 90% of the available CPU time.

viewer.

The resource usage is measured at well defined moments in time, by means of events. The entire software is event-based. The event for resource measurement purposes can be fired by programming it at the desired point in the code, or by a user interface event, or by means of the Unix command line.

The resource usage is measured twice: before performing the use case under study and afterwards. The measurement results show both the changes in resource usage as well as the absolute numbers. The initialization often takes more time in the beginning, while in a steady running system no more initialization takes place. Normally the real measurement is preceded by a set of actions to bring the system in a kind of steady state.

Note that the budget definitions and the *Unix utilities* fit well together, by design. The types of memory budgeted are the same as the types of memory measured by the Unix utilities. The typically used Unix utilities are:

ps process status and resource usage per process

vmstat virtual memory statistics

kernel resource stats kernel specific resource usage

The *heap-viewer* shows the free and allocated memory blocks in different colors, comparable with the standard Windows disk defragmentation utilities.

class name	current nr of objects	deleted since t_{n-1}	created since t_{n-1}	heap memory usage
AsynchronousIO	0	-3	+3	
AttributeEntry	237	-1	+5	
BitMap	21	-4	+8	
BoundedFloatingPoint	1034	-3	+22	
BoundedInteger	684	-1	+9	
BtreeNode1	200	-3	+3	[819200]
BulkData	25	0	1	[8388608]
ButtonGadget	34	0	2	
ButtonStack	12	0	1	
ByteArray	156	-4	+12	[13252]

Figure 15.16: Example output of OIT (Object Instantiation Tracing) tool

The *Object Instantiation Tracing* (OIT) keeps track of all object instantiations and disposals. It provides an absolute count of all the objects and the change in the number of objectives relative to the previous measurement. The system is programmed with Objective-C. This language makes use of run-time environment, controlling the creation and deletion of objects and the associated housekeeping. The creation and deletion operations of this run-time environment were rerouted via a small piece of code that maintained the statistics per class of object instantiations and destructions. At the moment of a trigger this administration was saved in readable form. The few lines of code (and the little run time penalty) have paid many many times. The instantiation information gives an incredible insight in the internal working of the system.

The *Object Instantiation Tracing* also provided heap memory usage per class. This information could not be obtained automatically. At every place in the code where malloc and free was called some additional code was required to get this information. This instrumentation has not been completed entirely, instead the 80/20 rule was applied: the most intensive memory consumers were instrumented to cover circa 80% of the heap usage.

Figure 15.16 shows an example output of the OIT tool. Per class the current number of objects is shown, the number of deleted and created objects since the previous measurement and the amount of heap memory in use. The user of this tool knows the use case that is being measured. In this case, for example, the *next image* function. For this simple function 8 new BitMaps are allocated and 3 AsynchronousIO objects are created. The user of this tool compares this number with his expectation. This comparison provides more insight in design and

implementation.

	test / benchmark	what, why	accuracy	when
public	SpecInt (by suppliers)	CPU integer	coarse	new hardware
	Byte benchmark	computer platform performance OS, shell, file I/O	coarse	new hardware new OS release
self made	file I/O	file I/O throughput	medium	new hardware
	image processing	CPU, cache, memory as function of image, pixel size	accurate	new hardware
	Objective-C overhead	method call overhead memory overhead	accurate	initial
	socket, network	throughput CPU overhead	accurate	ad hoc
	data base	transaction overhead query behaviour	accurate	ad hoc
	load test	throughput, CPU, memory	accurate	regression

Figure 15.17: Overview of benchmarks and other measurement tools

Figure 15.17 shows an overview of the benchmarking and other measurement tools used during the design. The overview shows per tool what is measured and why, and how accurate the result is. It also shows when the tool is being used.

The Objective-C overhead measurements, to measure the method call overhead and the memory overhead caused by the underlying OO technology, is used only in the beginning. This data does not change significantly and scales reasonably with the hardware improvements.

A set of coarse benchmarking tools was used to characterize new hardware options, such as new workstations. These tools are publicly available and give a coarse indication of the hardware potential.

The application critical characterization is measured by more dedicated tools, such as the image processing benchmark, which runs all the algorithms with different image and pixel sizes. This tool is home made, because it uses the actual image processing library used in the product. The outcome of these measurements were used to make design optimizations, both in the library itself as well as in the use of the library.

Critical system functionality is measured by dedicated measurement tools, which isolate the desired functionality, such as file I/O, socket, networking and database.

The complete system is put under load conditions, by continuously importing and exporting data and storing and retrieving data. This load test was used as

regression test, giving a good insight in the system throughput and in the memory and CPU usage.

15.7 Conclusion

This chapter described several decompositions: a functional decomposition of the image processing pipeline, a construction decomposition in layers and a process decomposition of the software. The image quality, throughput and response time have been discussed and especially the design choices that have been made to achieve the desired performance level. The design considerations show that design choices are related to consequences in multiple qualities and multiple CAFCR views. Reasoning over multiple CAFCR views and multiple qualities is needed to find an acceptable design solution. All information presented here was explicitly available in product creation documentation.

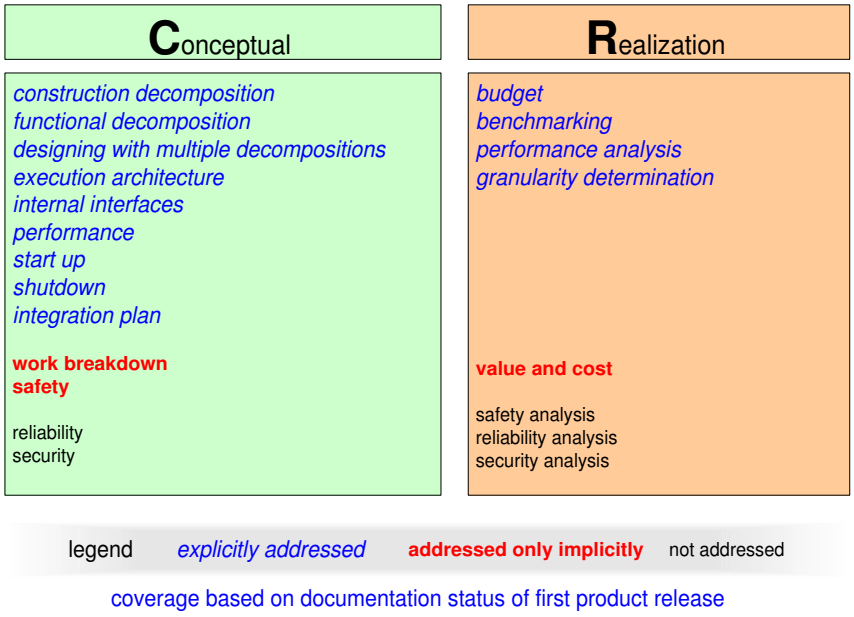


Figure 15.18: Coverage of submethods of the CR views

A number of submethods has not been described here, such as start up and shutdown, but these aspects are covered by the documentation of 1992. Figure 15.18 shows the coverage of the submethods described in part II by the docu-

mentation of the first release. This coverage is high for most submethods. Safety, reliability and security were not covered by the documentation in 1992, but these aspects were added in later releases of the product.

Chapter 16

Story Telling in Medical Imaging

16.1 Introduction

Stories have not been used explicitly in the development of the medical imaging workstation. Informally however a few stories were quite dominant in creating insight and focus. These informal stories do not meet all criteria described in Chapter 11, especially the specificity is missing. The typical case, as described in Chapter 14 is complementary to the stories. We now add the required specific quantitative details.

The main stories dominating the development were:

The sales story how to capture the interest of the radiologist for the product, see Section 16.2.

The radiologist at work describing the way a radiologist works. This story explains why the radiologist is **not** interested in viewing, but very interested in films, see Section 16.3.

The gastro intestinal examination how the URF system is used to examine patients with gastro intestinal problems. This story is not described here, because it is outside the scope of the discussed thread of reasoning

Section 16.4 relates the stories to the CAFCR model and discusses the criteria for stories as described in Chapter 11.

16.2 The Sales Story

The main function of the medical imaging workstation is rather invisible: layout and rendering of the medical images on film. To support the sales of the product more attractive appealing functionality was needed. The medical community is a rather conservative community, as far as technology is concerned: computers and software are mostly outside their scope. The sales approach was to provide an easy to use product, showing recognizable clinical information.

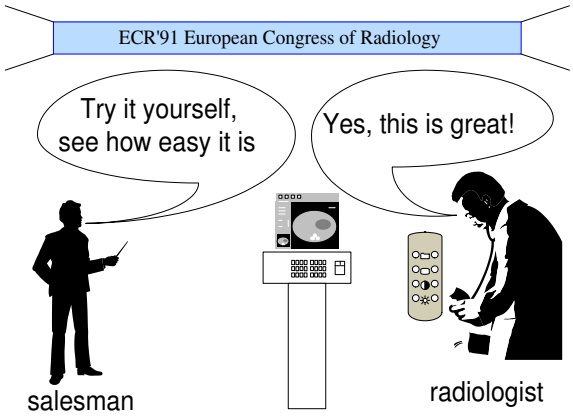


Figure 16.1: The main sales feature is easy viewing

At the European Congress of Radiology the system was shown to the radiologist. The radiologists were immediately challenged to operate the system themselves, see Figure 16.1.

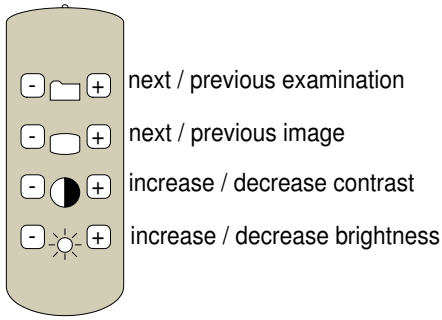


Figure 16.2: The simple remote control makes the viewing easy

The frequently used operations were available as single button operations on the remote control, see Figure 16.2: Select the examination, by means of previous/next examination buttons; Select image by previous/next image buttons; Adapt contrast and brightness by increase/decrease buttons.

Note that this is a nice sales feature, but that in day-to-day life the radiologist does not have the time to stand behind the workstation and view the images in this way. The viewing as described in Section 16.3 is much faster and efficient.

16.3 The Radiologist at Work

The radiologist has the following activities that are directly related to the diagnosis of a patient: supervising the examination, viewing the images to arrive at a diagnosis, dictating a report and verifying and authorizing the textual version of the report. Figure 16.3 shows these activities.

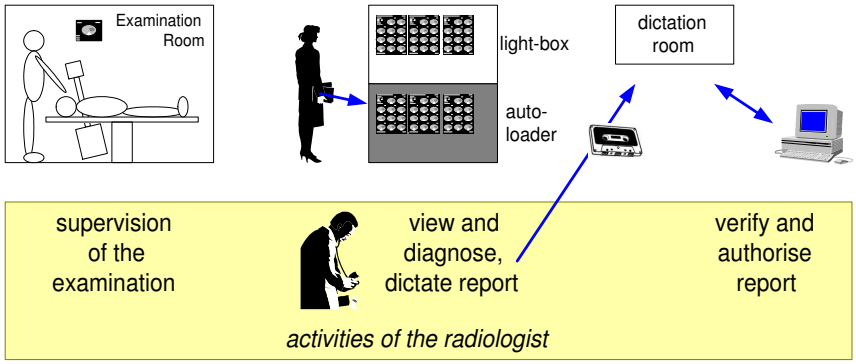


Figure 16.3: Radiologist work-spots and activities

The radiologist is responsible for the image acquisition in the examination room. The radiologist is not full-time present in the examination rooms, but supervises the work in multiple rooms. The radio technicians and other clinical personnel do most of the patient handling and system operation.

The films with examinations to be viewed are collected by clinical personnel and these films are attached in the right order to carriers in the auto-loader. The auto-loader is a simple mechanical device that can lift a set of films out of the store to the front of the lightbox. Pressing the button removes the current set of films and retrieves the next set of films.

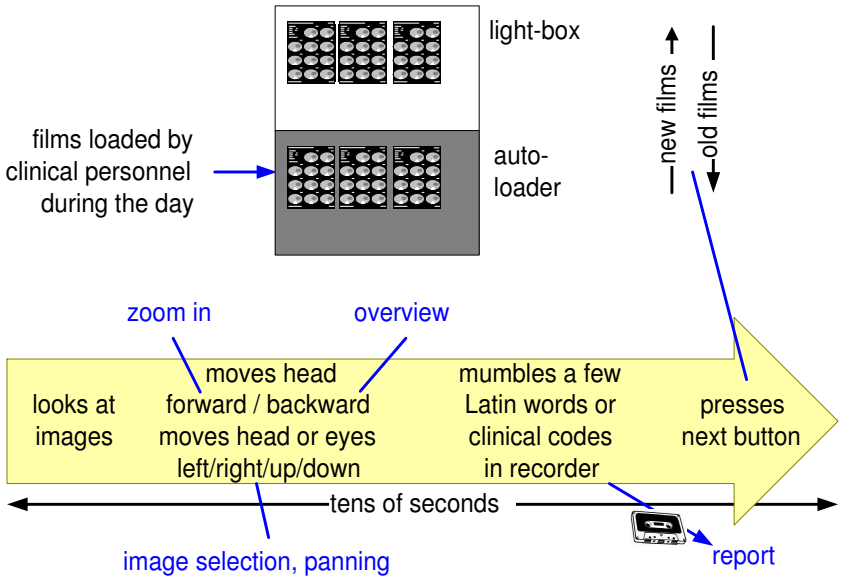


Figure 16.4: Diagnosis in tens of seconds

The activity of viewing and determining the diagnosis takes an amazingly short time. Figure 16.4 shows this activity in some more detail. A few movements of the head and eyes are sufficient to get the overview and to zoom in on the relevant images and the relevant details. The spoken report consists of a patient identification, a few words in Latin and or some standard medical codes. The recorded spoken report is sent to the dictation department; the transcription will be verified later. The radiologist switches to the next examination with a single push on the next button of the auto-loader. This entire activity is finished within tens of seconds.

The radiologist performs this diagnosis sometimes in between patients, but often he handles a batch of patient data in one session. Later on the day the radiologist will verify and authorize the transcribed reports, again mostly in batches.

16.4 Towards Design

The sales story provides a lot of focus for the user interface design and especially the remote control. The functions to be available directly are defined in the story. Implicit in this story is that the performance of these functions is critical, a poor

performance would kill the sales. The performance was not specified explicitly. However the implied response times were 1 second for image retrieval and 0.1 seconds for a contrast/brightness change. These requirements have a direct effect on the pipeline design and the user interface design.

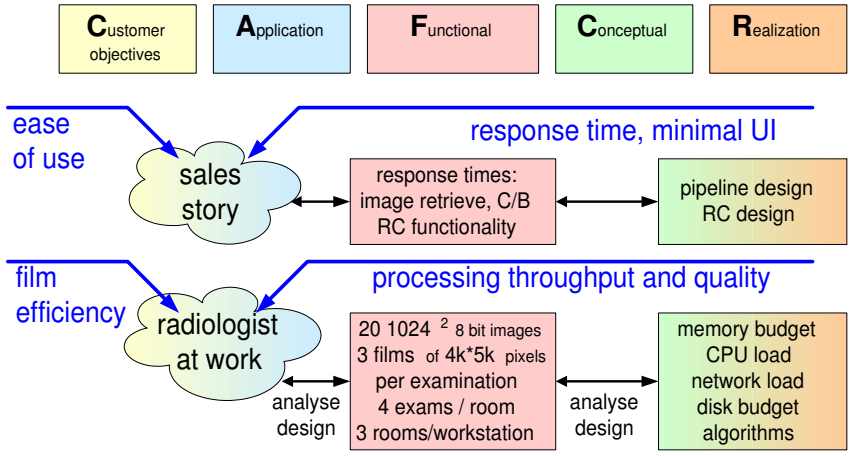


Figure 16.5: The stories in relation to the CAFCR views and the derived requirements and design choices

Figure 16.5 shows the flow from both stories to requirements and design. It also shows the inputs that went into the stories: at the commercial side the *ease of use* as sales feature and the *film efficiency* as the main application value. The gain in film efficiency is 20% to 50% relative to the screen copy approach used originally, or in other words the typical use of 3 to 5 film sheets is reduced to 2 to 3 film sheets. These numbers are based on the typical case described in Section 14.3.

The a priori know-how that the response time in a *software only* solution would be difficult, makes this a challenging story. The technical challenge in this story is to achieve the desired image quality and throughput, also in the *software only* solution.

The minimal user interface is also a design challenge. Without the sales story the user interface provided would have been much too technical, an overwhelming amount of technical possibilities would have been offered, without understanding the clinical world view.

The story of the radiologist at work, in combination with the typical case, is the direct input for the throughput specification. The throughput specification is used for the memory and disk budgets and the CPU and network loads. The

image quality requirements, in combination with the loads and budgets, result in algorithmic choices.

The original software completely ignored the need for printing images on film, it was not even present! The developer crew assumed that radiologists would use the workstation for "soft" diagnosis. Soft diagnosis is diagnosis from the monitor screen instead of film. A better understanding of the radiologist was needed to get the focus on the film printing functionality. The story immediately clarifies the importance of film sheets for diagnosis. The story also provides input for the functionality to create the layout of images and text on film. The auto-print functionality has been added in an extremely pragmatic way, by (mis-)using examination data fields to request printing. This pragmatic choice could only be justified by the value of this function as was made clear in this story.

16.5 Conclusion

Stories have not been used explicitly in the case. Somewhat less specific oral stories were provided by the marketing manager. Quantitative information was described in a typical case. The facts for quantification were provided by application managers. The presence of a quantified typical case provided the means for design, analysis and testing. The lack of explicit story, in combination with the poor coverage of the *Customer Objectives* and *Application* views as described in Chapter 14 in general, caused the late addition of the printing functionality.

Chapter 17

Threads of Reasoning in the Medical Imaging Case

17.1 Introduction

The thread of reasoning has not been applied consciously during the development of the Medical Imaging Workstation. This chapter describes a reconstruction of the reasoning as it has taken place. In Section 17.2 the outline of the thread is explained. Section 17.3 describes the 5 phases as defined in Chapter 12:

1. Select starting point (17.3.1)
2. Create insight (17.3.2)
3. Deepen insight (17.3.3)
4. Broaden insight (17.3.4)
5. Define and extend the thread (17.3.5)

17.2 Example Thread

Figure 17.1 shows a set of interrelated customer objectives up to interrelated design decisions. This set of interrelated objectives, specification issues and concepts is a dominant thread of reasoning in the development of the medical imaging workstation.

The objectives of the radiologist are at the same time reenforcing and (somewhat) conflicting. To achieve a good diagnostic quality sufficient time is required

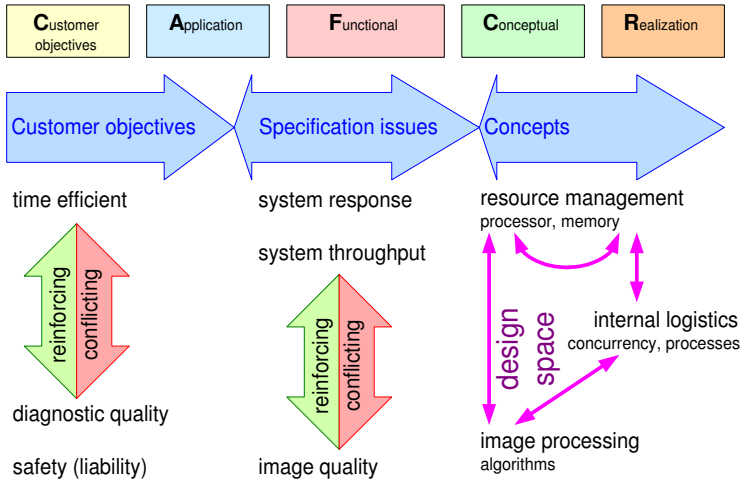


Figure 17.1: The thread of reasoning about the tension between time efficiency on the one hand and diagnostic quality, safety, and liability on the other hand. In the design space this tension is reflected by many possible design trade-offs.

or examine and study the results, which can be in conflict with time efficiency. On the other hand a good diagnostic quality will limit discussions and repeated examinations later on, by which good diagnostic quality can help to achieve time efficiency.

The customer objectives are translated into specifications. The diagnostic quality and safety/liability translate for example into image quality specifications (resolution, contrast, artefact level). A limited image quality is a primary source of a poor diagnostic quality. Artifacts can result in erroneous diagnostics, with its safety and liability consequences.

The time efficiency is achieved by system throughput. The workstation should not be the bottleneck in the total department flow or in the system response time. Waiting for results is clearly not time efficient.

Also at the specification level the reenforcing and the conflicting requirements are present. If the image quality is good, no tricky additional functions are needed to achieve the diagnostic quality. For instance if the image has a good clinical contrast to noise ratio, then no artificial contrast enhancements have to be applied. Function bloating is a primary source of decreased throughput and increased response times. The conflicting aspect is that some image quality functions are inherently time consuming and threaten the throughput and response time.

The design space is full of concepts, where design choices are needed. The concepts of *resource management*, *internal logistics* and *image processing algorithms* have a large impact on the system *response time* and *throughput*. The *image processing algorithms* determine the resulting *image quality*.

The design space is not a simple multi-dimensional space, with orthogonal, independent dimensions. The image processing algorithm has impact on the CPU usage, cache efficiency, memory usage, and image quality. The implementation of these algorithms can be optimized to one or two of these entities, often at the cost of the remaining optimization criteria. For instance: images can be stored completely in memory, which is most efficient for CPU processing time. An alternative is to store and process small parts of the image (lines) at a time, which is more flexible with respect to memory (less fragmentation), but the additional indirection of addressing the image line costs CPU time.

Adding concurrency partially helps to improve response times. Waiting times, for instance for disk reads, can then be used to do other useful processing. On the other hand additional overhead in context switching, and locking is caused by the concurrency.

The essence of the thread of reasoning is to have sufficient insight in the customer and application needs, so that the problem space becomes sharply defined and understood. This understanding is used to select the *sweet* spots of the design space, that satisfy the needs. Understanding of the design space is needed to sharpen the understanding of the problem space; in other words iteration between problem and solution space is required.

17.3 Exploration of Problems and Solutions

In this section the thread of reasoning is shown as it emerges over time. For every phase the CAFCR views are annotated with relevant subjects in that phase and the relations between the subjects.

Figures 17.2 to 17.6, described in Subsections 17.3.1 to 17.3.5, show the phases as described in Chapter 12. The figures show the main issues under discussion as dots. The relations between the issues are shown as lines between the issues, where the thickness of the line indicates the relative weight of the relationship. The core of the reasoning is indicated as a thick arrow. The cluster of issues at the start point and at the finish are shown as letter in a white ellipse. Some clusters of issues at turning points in the reasoning are also indicated as white ellipse.

17.3.1 Phase 1: Introvert View

At the moment that the architect (me) joined the product development a lot of technology exploration had been transformed into a working prototype, the so-called *basic application*. Main ingredients were the use of Object-Oriented (OO) technology and the vision that a "software only" product was feasible en beneficial.

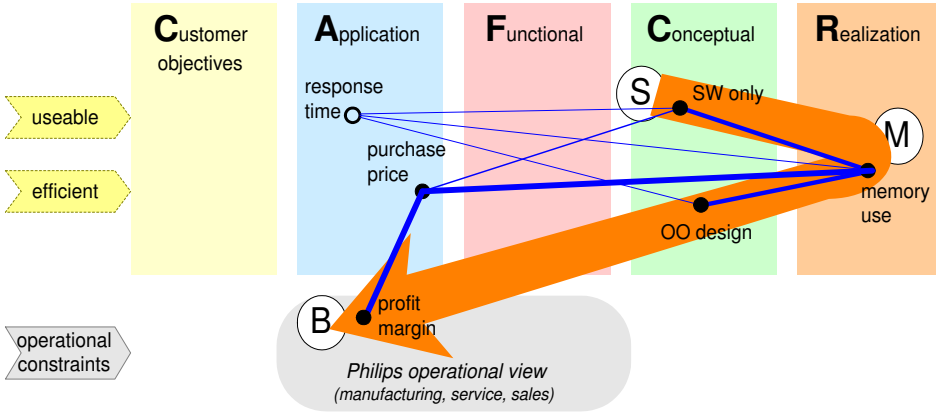


Figure 17.2: Thread of reasoning; introvert phase. The starting point (S) is the a priori design choice for a SW only solution based on Object Orientation. The consequence for resource usage, especially memory (M) and the business (B), especially product margin are explored.

Experienced architects will address two major concerns immediately: will the design with these new technologies fit in the technical constraints, especially memory in this case, and will the product fit in the business constraints (do we make sufficient margin and profit)?

The response time has been touched only very lightly. The system was only capable of viewing, an activity for which response time is crucial. The prototype showed acceptable performance, so not much time was spent on this issue. Design changes to eventually solve cost or memory issues potentially lower the performance, in which case response time can suddenly become important.

Figure 17.2 shows the thread of reasoning in this early stage. Striking is the introvert nature of this reasoning: internal design choices and Philips internal needs dominate. The implicitly addressed qualities are useability and efficiency. Most

attention was for the operational constraints. The direction of the reasoning during this phase has been from the Conceptual and Realization views towards the operational consequences: starting at the designers choice for OO and software only (S), via concerns over memory constraints (M) towards the business (B) constraints margin and profit. The figure indicates that more issues have been touched in the reasoning, such as response time from user point of view. In the day to day situation many more related issues have been touched, but these issues had less impact on the overall reasoning.

17.3.2 Phase 2: Exploring Memory Needs

The first phase indicated that the memory use was unknown and unpredictable. It was decided to extend the implementation with measurement provisions, such as memory usage. The OIT in the dynamic run time environment enabled a very elegant way of tracing object instantiations. At the same time a new concern popped up: what is the overhead cost induced by the run time environment?

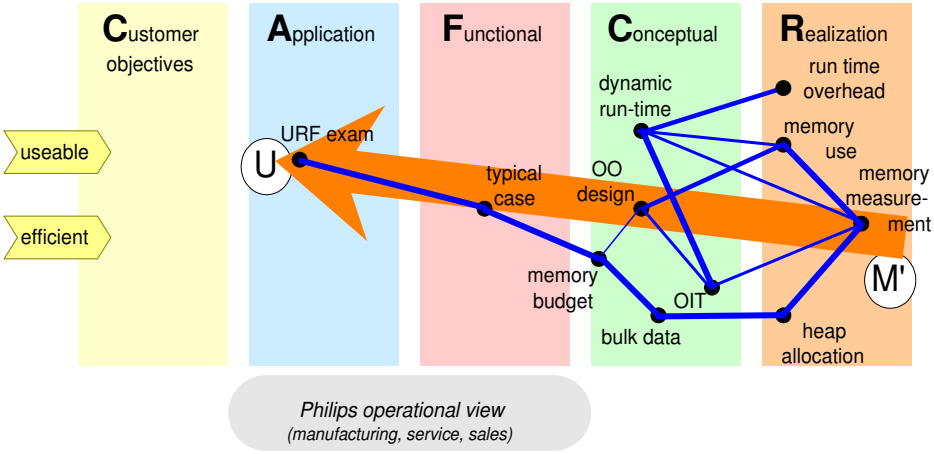


Figure 17.3: Thread of reasoning; memory needs. Create insight by zooming in on memory management (M'). Requirements for the memory management design are needed, resulting in an exploration of the typical URF examination (U).

The object instantiation tracing could easily be extended to show the amount

of memory allocated for the object structures. The large data elements, such as images, are allocated on the heap and required additional instrumentation. Via the bulkdata concept this type of memory use was instrumented. Bottom up the insight in memory use was emerging.

The need arose to define relevant cases to be measured and to be used as the basis for a memory budget. An URF examination was used to define a typical case. Now the application knowledge starts to enter the reasoning process, and the reasoning starts to become more extrovert. Efficiency and usability are the main qualities addressed.

Figure 17.3 shows the thread of reasoning for Phase 2. The reasoning is still bottom-up from Realization towards Application View. The realization concerns about speed and memory consumption (M') result into concepts for resource management and measurement support. For analysis and validation a use case description in the Functional view is needed. The use case is based on insight in a URF examination (U) from application viewpoint.

17.3.3 Phase 3: Extrovert View Uncovers Gaps in Conceptual and Realization Views

The discussion about the URF examination and the typical case made it very clear that radiologists perform their diagnoses by looking at the film on the lightbox. This is for them very efficient in time. Their speed of working is further increased by the autoloader, which quickly shows all films of the next examination.

To support this typical workflow the production of filmsheets and the throughput of films and examinations is important. Interactive viewing on the other hand is from the radiologist's point of view much less efficient. Diagnosis on the basis of film takes seconds, diagnosis by interactive viewing takes minutes. The auto-print functionality enables the production of films directly from the examination room.

auto-print functionality requires lots of new functions and concepts in the system, such as background printing (spooling), defining and using film layouts, using the right rendering, et cetera. The processing library must support these functions. Also an execution architecture is required to support the concurrency: server processes and spool processes are introduced. Last but not least, hardcopy units (HCU), for example laser printers, need to be interfaced to the system. A new set of components is introduced in the system to do the printing: hardcopy interface hardware, hardcopy driver, and the hardcopy units themselves.

During this phase the focus shifted from efficiency to effectiveness. Efficiency

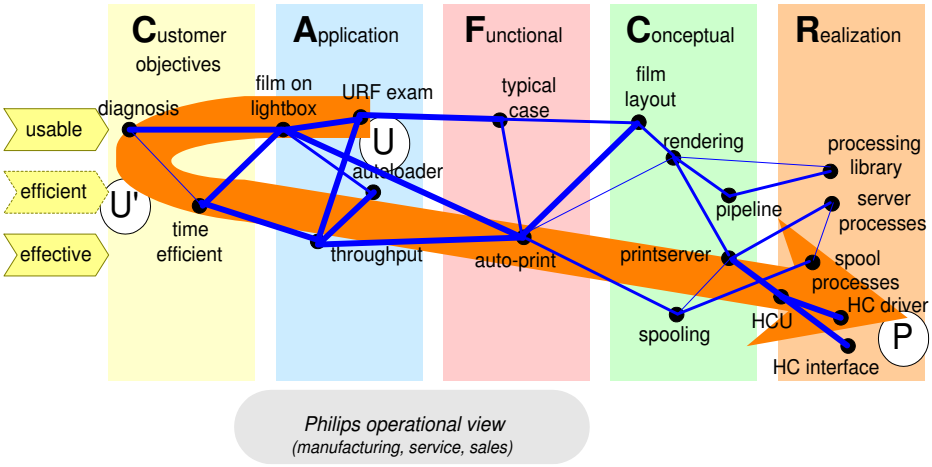


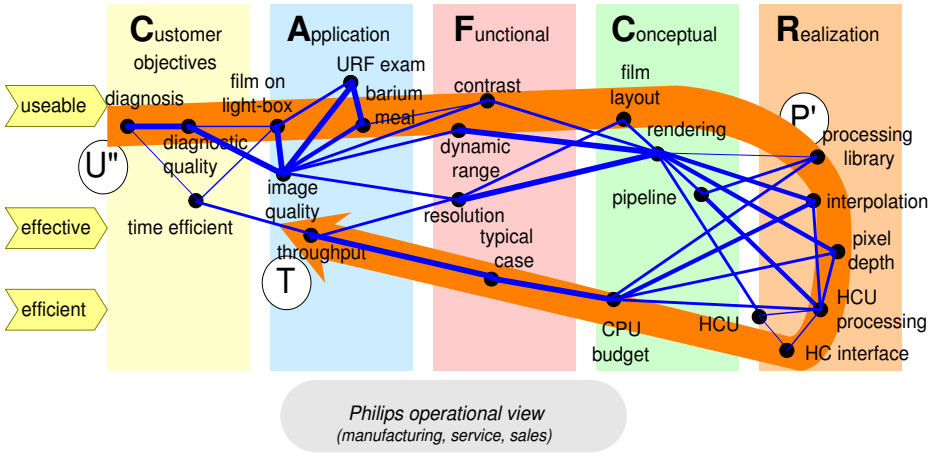
Figure 17.4: Thread of reasoning; uncovering gaps. The insight is deepened by further exploration of the URF examination (U) and the underlying objectives (U') of the radiologist. The auto-print functionality is specified as response for the radiologist needs. The technical consequences of the auto-print are explored, in this case the need for printing concepts and realization (P).

is mostly an introvert concern about resource constraints. Effectiveness is a more extrovert concern about the quality of the result. Hitchins clearly explains in [32] efficiency and effectiveness, and points out that the focus on efficiency alone creates vulnerable and sub-optimal systems. Usability remains important during this phase, for example auto-print.

Figure 17.4 shows the thread of reasoning of Phase 3. The insights obtained during the previous phase trigger a further exploration of the Customer Objectives and Application View. The insight that an efficient diagnosis (U') is performed by means of film sheets on a lightbox (U) triggers the addition of the auto-print function to the Functional View. New concepts and software functions are needed to realize the auto-print function (P). The direction of reasoning is now top-down over all the CAFCR views.

17.3.4 Phase 4: from Diagnosis to Throughput

The discussion about URF examinations and the diagnostic process triggers another thread, a thread about the desired diagnostic quality. The high brightness and resolution of films on a lightbox ensures that the actual viewing is not degrading the diagnostic quality. The inherent image quality of the acquired and printed image is critical for the final diagnostic quality.



from extrovert diagnostic quality, via image quality, algorithms and load, to extrovert throughput

Figure 17.5: Thread of reasoning; phase 4. The insight is broadened. Starting at the objective to perform *diagnosis* efficient in time (U"), the application is further explored: type of examination and type of images. The specification of the imaging needs (contrast, dynamic range, resolution) is improved. The consequences for rendering and film layout on a large set of realization aspects (P') is elaborated. The rendering implementation has impact on CPU usage and the throughput (T) of the typical case.

At specification level the image quality is specified in terms of resolution, contrast and dynamic range. At application level the contrast is increased by the use of barium meal, which takes the contrast to the required level in these soft (for X-ray low contrast) tissues. At the same time the combination of X-ray settings and barium meals increases the dynamic range of the produced images.

The size of the images depends on the required resolution, which also de-

termines the film layout. The rendering algorithms must fulfil the image quality specifications. The rendering is implemented as a pipeline of processing steps from an optimized processing library.

One of the costly operations is the interpolation. One of the design options was to use the processing in the hardcopy unit. This would greatly relieve the resource (processor and memory) needs, but it would at the same time be much less flexible with respect to rendering. It was decided not to use the hardcopy unit processing.

A CPU budget was created, based on the typical case and taking into account all previous design know-how. This CPU budget did fit in the required throughput needs.

Usability, effectiveness and efficiency are more or less balanced at this moment.

Figure 17.5 shows the thread of reasoning for Phase 4. During this phase the reasoning iterates over all the CAFCR views. The diagnostic quality (U") in the Customer Objectives View results via the clinical acquisition methods in the Application view in image quality requirements in the Functional View. The layout and rendering in the Conceptual view result in a large set of processing functions (P') in the Realization view. The specific know how of the processing in the Realization is used for the CPU and memory budgets in the conceptual view, to validate the feasibility of supporting the typical case in the Functional view. The typical case is a translation of the throughput (T) needs in the Application View.

17.3.5 Phase 5: Cost Revisited

At this moment much more information was available about the relation between resource needs and system performance. The business policy was to use standard of-the-shelf workstations. The purchase price by the customer could only be met by using the lowest cost version of the workstation. Another policy was to use a Philips medical console, which was to be common among all products. This console was about half of the material cost of the Medical Imaging workstation.

The real customer interest is to have a system that is economically sound, and where throughput and cost of ownership (CoO) are balanced. Of course the main clinical function, diagnosis, must not suffer from cost optimizations. A detailed and deep understanding of the image quality needs of the customer is needed to make an optimized design.

Note that at this moment in time many of the considerations discussed in the

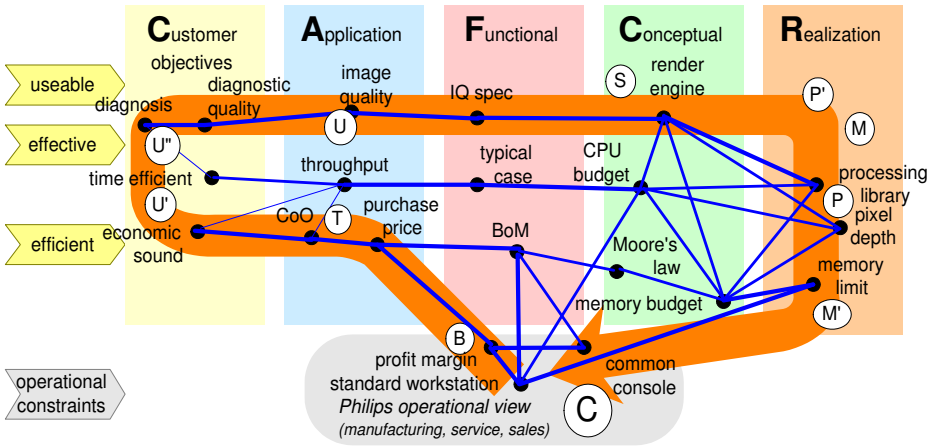


Figure 17.6: Thread of reasoning; cost revisited. The entire scope of the thread of reasoning is now visible. Sufficient insight is obtained to return to the original business concern of margin and cost (C). In the mean time additional assumptions have surfaced: a common console and standard workstation to reduce costs. From this starting point all other viewpoints are revisited: via time efficient diagnosis to image quality to rendering and processing and back to the memory design.

previous steps are still valid and present. However Figure 17.6 is simplified by leaving out many of these considerations.

Besides efficiency, effectiveness, and usability, the operational constraint is back in the main reasoning thread. At this moment in time that makes a lot of sense, because problem and solution space are sufficiently understood. These constraints never disappeared completely, but the other qualities were more dominant in the intermediate phases.

Figure 17.6 shows the thread of reasoning in Phase 5. The original business viewpoint is revisited: do we have a commercial feasible product? A full iteration over all CAFCR views relates product costs (C) to the key drivers in the Customer Objectives. The main tensions in the product specification are balanced: image quality, throughput of the typical case and product cost. To do this balancing the main design choices in the Conceptual and Realization views have to be reviewed.

17.4 Conclusion

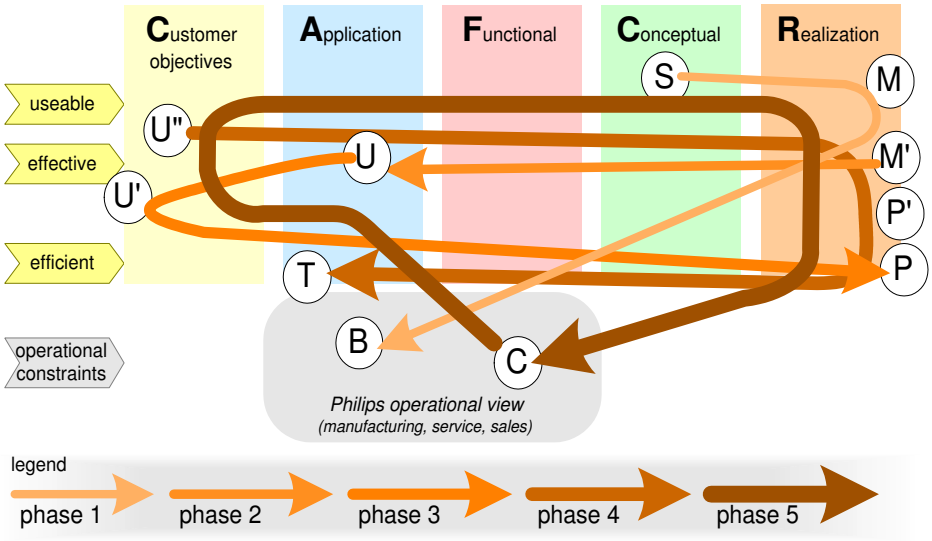


Figure 17.7: All steps superimposed in one diagram. The iterative nature of the reasoning is visible: the same aspects are explored multiple times, coming from different directions. It also shows that jumps are made during the reasoning.

The know-how at the start of the product creation was limited to a number of nice technology options and a number of business and application opportunities. The designers had the technology know-how, the marketing and application managers had the customer know-how. The product creation team went through several learning phases. Figure 17.7 shows the many iterations in the five phases. During those phases some of the know-how was shared and a lot of new know-how emerged. The sharing of know-how made it possible to relate customer needs to design and implementation options. The interaction between the team members and the search for relations between needs and designs triggered many new questions. The answers to these questions created new know-how.

The architecting process has been analyzed in retrospect, resulting in this description of *threads of reasoning*. This Chapter *Threads of Reasoning* shows that:

- The specification and design issues that are discussed fit in all CAFCR views or the operational view.

- The positioning of the issues and their relationships in the CAFCR views enable a compact description of the reasoning during the product creation.
- Submethods are used to address one issue or a small cluster of issues.
- Qualities are useful as integrating elements over the CAFCR views.
- The *threads of reasoning* are an explicit way to facilitate the interaction and the search for relations.
- The *threads of reasoning* create an integral overview.
- The *threads of reasoning* facilitate a converging specification and design.

Part IV

Evaluation, Discussion and Conclusions

Chapters in Part IV:

Chapter 18

Evaluation of the Architecting Method

18.1 Introduction

The hypothesis formulated in Chapter 6 is evaluated by means of the Medical Imaging Workstation case. This evaluation uses the criteria that have been defined in Chapter 6.

Figure 18.1 shows how the case maps on the hypothesis and repeats the criteria. This figure is used as the basis for the evaluation of the case.

The hypothesis narrows the scope: the method is applied on *software and technology intensive complex systems* and the product creation takes place in *heterogeneous environments*. The medical imaging workstation fully fits in these constraints. The system requires many hardware and software technologies, see Figure 3.5 and the list of technology innovations in Section 3.2. Also a lot of physics and clinical application related technologies are required to translate pixels into clinically relevant information. The heterogeneity of the environment is present in multiple dimensions: *modalities* (X-ray, CT, MRI), *applications* (gastro-intestinal, vascular, CT and many more) and *releases* (URF, vascular, CT, and MRI; each of them interoperable with multiple releases).

Section 18.2 evaluates the design of the product. The design quality is a measure for the sustainability of the product and for the support of the method to the PCP team. Section 18.3 evaluates the product itself. The product evaluation addresses the criteria 1 and 2, of Figure 18.1, which address the *commercial* success. Section 18.4 discusses how the *method* has been instrumental in creating the

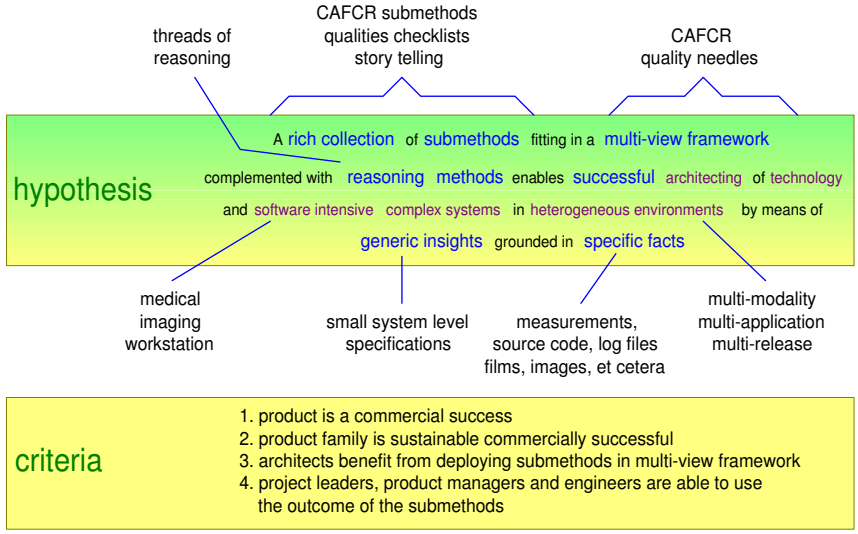


Figure 18.1: Annotated hypothesis and criteria as basis for the evaluation

product and how it has contributed to sustainable success, thus evaluating criterion 3: Criterion 4, the usability of the outcome, is evaluated in Section 18.5. The conclusion of the complete evaluation is articulated in Section 18.6.

18.2 Design Evaluation

Figure 18.2 shows an evaluation of the design quality, which is based on an internal technology improvement plan [50].

The community of designers was very content about many of the concepts and implementations. This assessment is rather subjective, because the designers are creators and users of the design at the same time. The following three questions are a somewhat more objective way to make the quality assessment more explicit:

- Does the product, based on this design, fulfill the requirements?
- How does the design support ongoing developments? How is development effort (in time and manpower) affected when new functionality is added, for instance as shown in Figure 14.10?
- Benchmarking: How does the product and design compare with other products? For example, compare code size in relation to the offered functionality

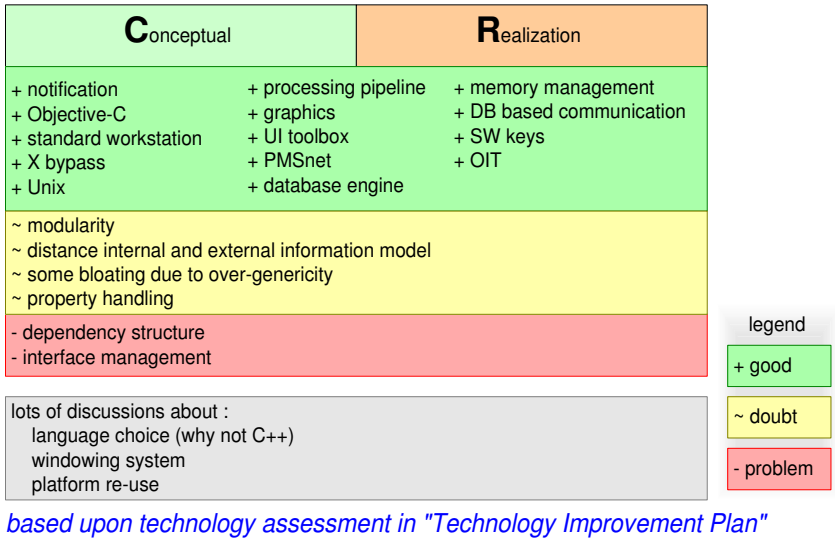


Figure 18.2: Evaluation of the design of the medical imaging workstation

for multiple products.

Most of the success of the first product and its successors, as described in Section 18.3, are due to the power, flexibility, expressiveness, and richness of the concepts and implementations mentioned in the “good”-section of Figure 18.2.

Note that most of the strong point are concepts that have been chosen and implemented by the software designers rather than by the architect. The SW designers deserve the credit for the selection and implementation of these concepts. The architect is involved in the selection and deployment of the concepts, but more supportive and monitoring than leading. The submethods that have been presented do not deal with the selection of these concepts, although they provide some means such as criteria. The submethods mostly provide means at higher integration levels.

A *code size comparison* is hard to reconstruct at this moment, mostly because it requires an objective measure of functionality. The 360kloc (including comments) of code in the medical imaging workstation divides about equally into code for viewing functionality and code for the modality products. However the workstation removed many of the constraints of the modality products. It allowed, for example, arbitrary sized images instead of 512^2 and 1024^2 images only. In other words more functionality is offered in about the same amount of code.

The designers who actually used the concepts and the implementation were very positive. People outside of the medical imaging workstation product group, however, had a lot of critical remarks about the design and implementation. The feedback of this group was not based on practical experience but rather from documentation (best case) or hearsay (worst case). This situation was caused by the political climate within the division, where competition for funding caused a lot of this kind of non-rational arguments.

The critical remarks of the group of outsiders were taken seriously. As a result a technology improvement plan was written, on which figure 18.2 is based. The aspects assessed as "doubtful" and negative are discussed below.

The first phases of the improvement plan focused on improvements of the *modularity*, *interface management*, *dependency structure*, and the *distance between internal and external information model*. The problem with modularity was that only fine-grain modularity, at class or file level, existed. No higher aggregation level existed and no explicit dependency structure was defined. In general the class level granularity was good. Some classes, however, were too generic, which resulted in bloating. Inheritance was used too much, which often happens when development teams start to use OO techniques.

The internal and external data models were two entirely different entities managed by different people. From an interoperability point of view this is a risk. This risk becomes more prominent when different modalities have to interoperate.

The delayed design choices, and data about installation, configuration, and customization were all stored in property files. This mechanism did not scale up well and the information was not recognized as regular code. The risk is comparable with missing source code management. Another risk is loss of overview. Note that those risks were not yet acute in the first product with its limited configurability and functionality.

The interface management was one of the main issues of this design. The product did not suffer from any interface management problem, but follow-on products, which re-used the code base, complained a lot. All changes applied somewhere in the code could propagate to other parts of the code.

In the organization a lot of noise was present in discussions about the programming language (Objective-C [5]). The opinion of managers and engineers outside of the project was that Objective-C was a non-standard language, which was dying due to lack of followers in the rest of the world. On the other hand, a lot of the valuable concepts built upon the dynamic nature of Objective-C. These concepts would have bloated significantly if C++ would have been used for the implementation. In retrospect, a language disappears not as fast as predicted by

many of its opponents. Somewhat less heated was the debate about the windowing system. Most organizational noise was removed by the use of the X-compatible bypass (Nix).

A lot of disturbance was caused by the platform expectations in the organization. The original objective of the development team [52] was to create a re-usable platform for Philips Medical Systems. The team focus changed to create medical imaging workstation products. From the product management point of view the platform creation became a non-issue: platforms are not sold to external customers! The product groups in the context did not follow this focus change. They expected a cost reduction to be achieved by sharing the development costs of a viewing platform.

18.3 Product Evaluation

The case describes the development of the first product release of the Medical Imaging Workstation. Figure 18.3 shows the strong and weak points, as they were perceived at the outside of the product.

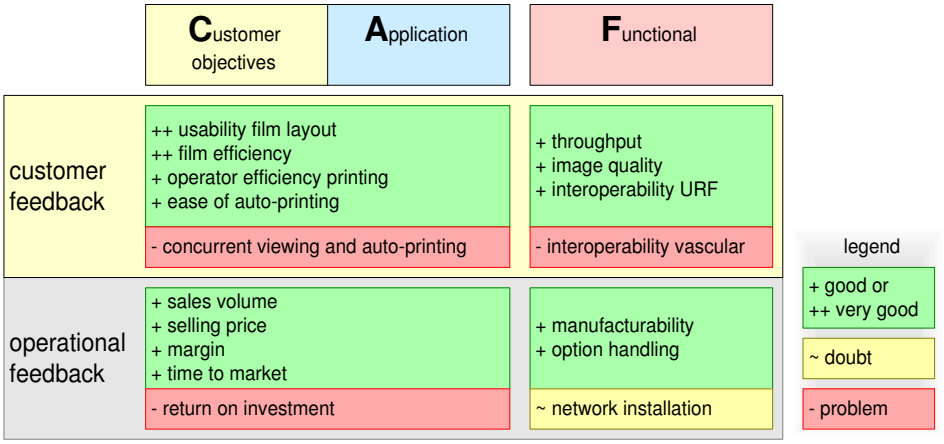


Figure 18.3: Evaluation of the medical imaging workstation product

The feedback from the URF customers, obtained via application managers who visit many URF departments, was very positive about the *usability* of the *film layout* and the *film efficiency*. Also the *operator efficiency of printing* and *ease of auto-printing* was appreciated. Viewing was used only very limited. The few users who used it also for viewing complained about the simultaneous performance of

viewing and auto-printing. Note that this was no surprise; it was even according to specification. It indicates that to a small subset of the customers the specification was insufficient.

At specification level the *throughput*, *image quality*, and *interoperability with URF systems* were according to specification. The specification is appreciated by the customer. The interoperability with vascular systems was below internal (product management) expectations. Product management expected vascular systems to interoperate in the same way as URF systems. However, vascular systems are used quite differently than URF systems. The functionality and performance needs of vascular surgeons differ from the radiologist's needs. The system did not provide significant value for vascular use. The vascular product group did not have a clear incentive to improve the interoperability, because of the lack of added value, Not many customer complaints were filed about this problem[1].

From a business point of view the product was clearly a success. It yielded a good *sales volume* (increasing from circa 50 systems in the first year to hundreds of systems in later years), the right *selling price*, a reasonable *margin*, and a good *time to market*. The return on investment (ROI)[2], however, only occurred after several years.

At the portfolio level of Philips Medical Systems (PMS) the turnover of these workstations was negligible. Modality systems (MRI, X-ray, CT-scanners) form the bulk of the turnover. Modality systems have a limited innovation speed, due to their multi-technology complexity and their safety requirements. The medical imaging workstation provided a means to introduce *image handling* innovation faster, because this product has its own (more dynamic) life-cycle. In later years the strategic value of an independent platform for image handling innovation at PMS level turned out to be large. Additional modality[3] turnover could be realized thanks to the image handling capabilities of the medical imaging workstation.

Feedback from manufacturing and service departments indicates that the operational requirements, such as *manufacturability* and *option handling*, were fulfilled satisfactorily. Installation of networking functionality overlapped with other

[1] Service engineers and application managers file complaints of users by means of Problem Reports. A control board analyses and discusses incoming Problem Reports regularly.

[2] Product groups have a profit/loss responsibility. Losses are accepted for about 2 years, after such a start-up period a "normal" ROI is demanded. The volume of all products, including options, was after about 2 years at an acceptable level. Especially the (software) options helped to get at an acceptable ROI.

[3] The term *modality* system is used for image acquisition systems. The image acquisition technique, such as MR, CT, US, or X-ray, is dominant in the system design. This technique is called the modality.

hospital disciplines, such as IT departments and facility management. For the Philips service work force this was a new area. The networking was not a big issue, but it required quite a lot operational attention.

18.4 Evaluation of Architecting Method

Subsection 18.4.1 evaluates the use of the submethods, described in the Chapters 8 and 9 and applied in the Chapters 14 and 15. The use of the qualities and the quality checklist (Chapter 10) are evaluated in Subsection 18.4.2. Subsection 18.4.3 evaluates the story telling (Chapter 11 and 16). The reasoning methods (described in Chapter 12 and applied in Chapter 17) are evaluated in Subsection 18.4.4. The integration of all components of the methods is evaluated in Subsection 18.4.5.

18.4.1 CAFCR Submethods

Figure 18.4 shows the submethods per view. Every method is annotated with its usage in the case. Many submethods are used explicitly, although not always exactly as described in part II. The fact that these submethods have been used recognizes the need for these methods.

Some subjects covered by submethods have been discussed during the product creation, but did not result in any specific consolidation of data in specifications. In figure 18.4 these submethods are labelled *"only implicitly"*. Often the coverage of these subjects was rather partial. They have been discussed, but they were not thoroughly analyzed.

The fact, for instance, that safety has not been explicitly addressed in the first release is due to the low severity factor of this product. This product does not immediately endanger patients or operators. X-ray systems, on the other hand, use radiation and heavy moving parts that can be dangerous for patients as well as personnel. These systems take measures to cope with the risks, such as shielding and detection. In later releases of the medical imaging workstation the safety has been addressed explicitly, with a mandatory hazard analysis and accompanying specification and design measures.

The subjects that have not been covered in the first release were less important and critical than the covered subjects. This selection process is described in Figure 7.7. Some of these subjects, for instance security, have been addressed explicitly in a later release.

The most remarkable observation from Figure 18.4 is that the *Customer Objectives* and *Application* views are poorly covered. As discussed in Section 17.3

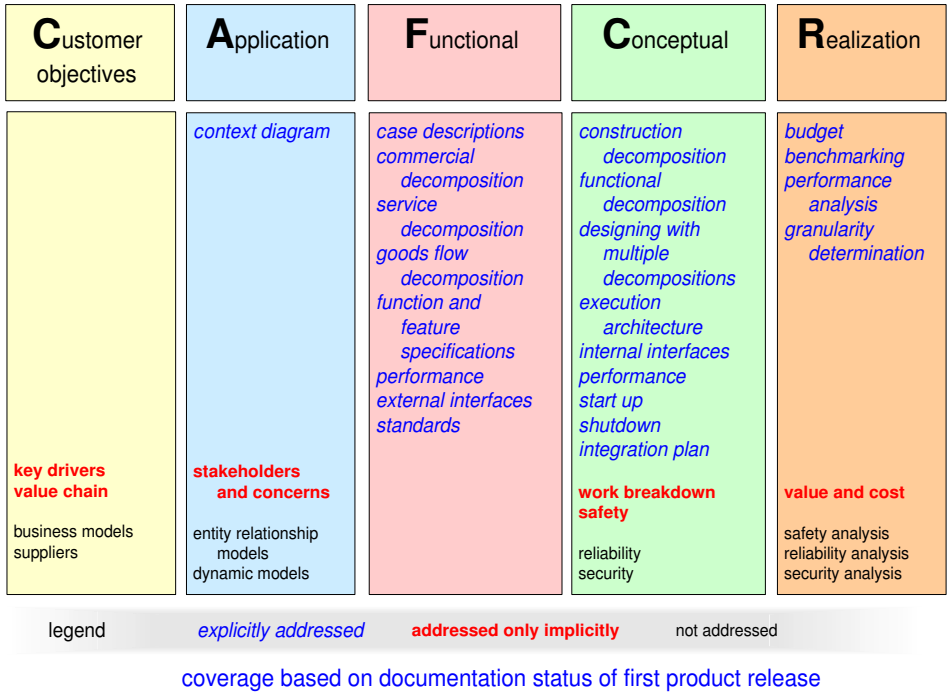


Figure 18.4: Coverage of submethods discussed in part II

the introvert culture of the development teams causes this unbalance.

18.4.2 Qualities

Figure 18.5 shows the use of qualities that appear in in the documentation structure in 1996, when the product family was already more mature. A significant amount of qualities were explicitly documented in separate documents: *image quality*, *safety*, *security*, *throughput or productivity*, *connectivity*, *resource utilization*, *configurability* and *installability*. Many other qualities were addressed as part of other documentation, such as functional requirement specifications or the technical product documentation.

Figure 18.6 shows the relative coverage of the qualities as they have been applied in the case. This coverage is determined by the amount of explicit and implicit information present in the system documentation, see Figure 18.5. The relative relevance of these qualities for this business is also shown. The relevance

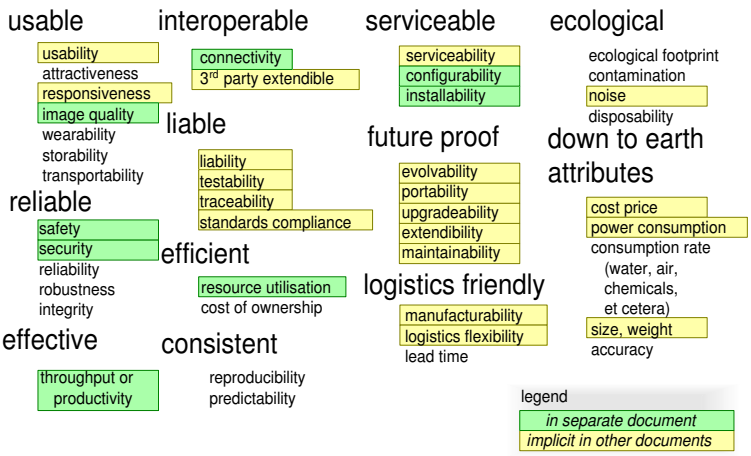


Figure 18.5: Quality documentation in 1996

is based on experience used in a retrospective assessment. The qualities in this figure have been aggregated. This aggregated format makes it immediately clear that only a subset of qualities is relevant per business.

For medical imaging qualities such as *ecological*, *logistics friendly* and *down to earth attributes* are relatively unimportant. In other systems, for instance printers, all of these qualities are important.

The conclusion of Figures 18.6 and 18.5 is that the qualities play an important role in the product creation. For in this product many qualities were even documented in separate documents.

18.4.3 Story Telling

Story telling has not been deployed in the medical imaging case in the way described in in Chapter 11. In the documentation of the system there is a document called *typical case*. This document partially fulfilled this role. The *typical case* in combination with the oral version, described in Chapter 16, proved to be very valuable to connect the clinical world and the engineering world. The radiologist story created the focus on printing on film. The efficient film production became the number one selling argument of the product.

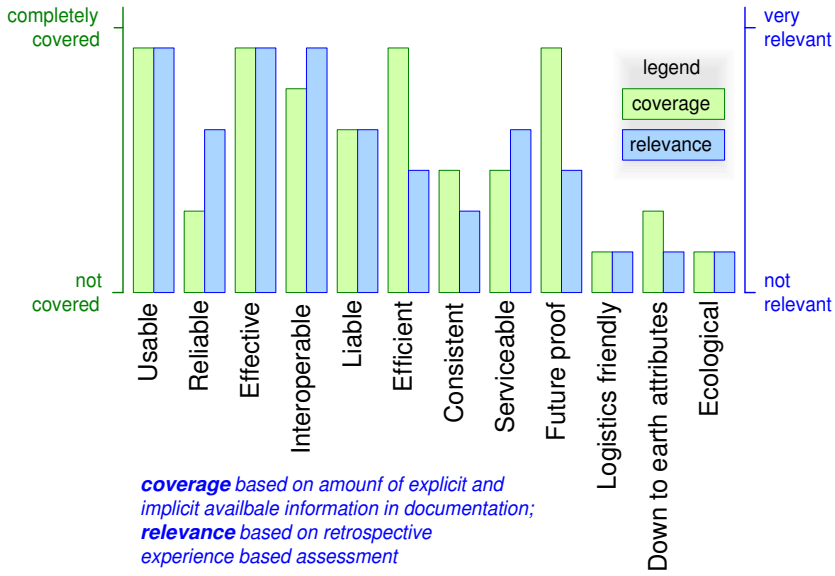


Figure 18.6: Coverage profile of qualities

18.4.4 Threads of Reasoning

The threads of reasoning were only semi-consciously applied. The fast iteration over multiple views was consciously applied. The integral understanding emerged as a result of this fast iteration. In retrospect this approach can be visualized as a *thread of reasoning* as described in Chapter 17. It is this integral understanding that helped to finish the product within the constraints. The integral understanding can indirectly be observed by the handling of the problems found during integration, see Sections 13.4 and 13.5.

The introduction date is a very important business constraint: late introduction means lower market share and lower margins. The integral understanding made it possible to finish the product in time. Without integral understanding it would have taken much more time (months or years) to get the product at an acceptable level.

18.4.5 The Integration of the Method

The hypothesis starts with "*A rich collection of submethods fitting in a multi-view framework complemented with reasoning methods*". In other words the *submeth-*

ods, the *multi-view framework* and the *reasoning methods* must be employable in a complementary fashion. The integration of components of the methods is evaluated in this subsection.

The main value provided by the architecting method is in detecting and solving **integral** problems, such as the *throughput*, *image quality*, and *memory utilization*. Most local aspects (single concepts, single functions) were engineered in a professional way. Integration problems arise across the boundaries of concepts and functions, and emerge due to the dynamic behavior of many components. The submethods and the framework add means to do the *system* level design.

Many submethods are based on very specific and detailed facts. The *typical case*, as described in Section 14.3, also focuses on a completely quantified description, which enables specific analysis and design actions. The specifications at the system level have to be rather generic to keep the specifications usable and the size manageable. Typical quality specifications, for instance, were between 4 and 8 pages of content. In other words, in the consolidation of the design there is very limited room for specific details.

The architect continuously operates in this field of force: the need for *genericity* in guidelines and rules, while the added value is mostly in the integral understanding emerging from a large amount of highly *specific detailed facts*.

In the medical imaging workstation case the amount of viewpoints, submethods, and qualities used was good: no dramatic problems caused by missing viewpoints, submethods or qualities arose later. All of the viewpoints, submethods, and qualities are highly relevant in solving the integration problems. The duration of the integration of the first product and of the later products, see Figure 14.10, was manageable. The relevance of the viewpoints, submethods, and qualities is shown in Chapter 17. The amount of method components was manageable for the architects, and the results could be absorbed by the stakeholders. The selection of the methods emerged by (unconsciously) applying the threads of reasoning. Conclusion: the reasoning method resulted in a minimal set of required components of the method, and the reasoning method supported the integration of these components.

The balance between genericity and specificity, which is formulated at the end of the hypothesis "*by means of generic insights grounded in specific facts*", did work out well in the case. Sufficient specific details, such as measurements, source code, log files, films, and images, were touched to form the basis for the emerging understanding. The size of the system level documentation was rather limited; the integral overview was clearly present in the documentation, the overview was not hidden in a vast amount of details. Some of the engineers, however, criticized the

abstraction level of the documentation. This criticism is discussed in Section 18.5, which evaluates the usability.

18.5 Usability Evaluation of the Outcome of the Architecting Method

Criterion 4 states "*project leaders, product managers and engineers are able to use the outcome of the submethods*". This section evaluates how the PCP team members have experienced the outcome of the submethods.

The results of the submethods in the CAFCR views are consolidated in product and design specifications. Figure 18.7 shows who uses the results, what the results are used for, and what complaints have been voiced by the engineers. These specifications have been used by a wide variety of stakeholders: *product management*, *application*, *project leaders*, *engineers*, *test engineers*, *purchasing*, *manufacturing*, and *suppliers* for a wide range of purposes: to *create detailed specifications*, to *test*, to *communicate*, to *derive documentation* such as manuals, and as basis for *succeeding products*.

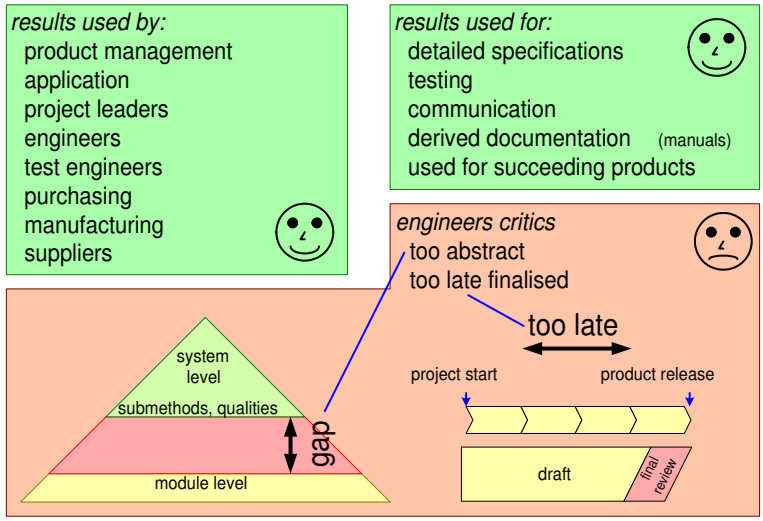


Figure 18.7: Users and usage of the results of the architecting method

Some engineers complained about the level of abstraction of the output. "How can I use this specification; I need more guidance". The problem is that the dis-

tance from system level specifications (high abstraction level) to module level implementation (low abstraction level) is very large (from tens of documents to hundreds of thousands lines of code). For some functional subsystems this gap was bridged by deploying the architecting method recursively. Some members of the product creation team were capable of doing this, but for other subsystems the gap never got bridged.

The solution for this class of problems is often outside the scope of the method. This gap problem, for instance, can be solved by working on the quality of the PCP team. To improve the team quality the skills of the individuals can be improved and the composition of the team can be changed to obtain the required distribution of skills in the team. The skills of the designers at the module level are increased to be able to work directly from the system level output. Nevertheless, this will always be an area of tension: problems that are complex by nature cannot be simplified infinitely. The quality of the PCP team is solved in the wider context of the PCP, as shown in Figure 1.2: people (skills), organization (team composition), process (methods).

Another complaint uttered by some of the engineers was that the results were frozen quite late. Many specifications exist for a long time in *draft* status, to get their final review somewhere in the integration phase. On the one hand this is a fair criticism: most engineers need stable information to make good progress. The problem on the other hand is that product creation in complex and innovative environments is full of uncertainties: is it feasible?, what do we need exactly? how much does it cost? And these questions are often mutually dependent. No method will ever be able to know the unknowns; in the best case it will help to cope with the unknowns. Support for unknowns makes it possible to discover unknown issues early. Another way to cope with the unknowns is to minimize vulnerability.

18.6 Conclusion

The case has been used to test the hypothesis. The results of the evaluation are summarized in Figure 18.8. In many aspects the criteria are met:

Design and Integration According to Section 6.4 the quality of the design and the integration is a measure for Criterion 3, and the quality of the design is a prerequisite for Criterion 1. The quality of design was good (Section 18.2). The quality of the integration was also good (Subsection 18.4.5).

Criterion 1 The product is a commercial success (Section 18.3).

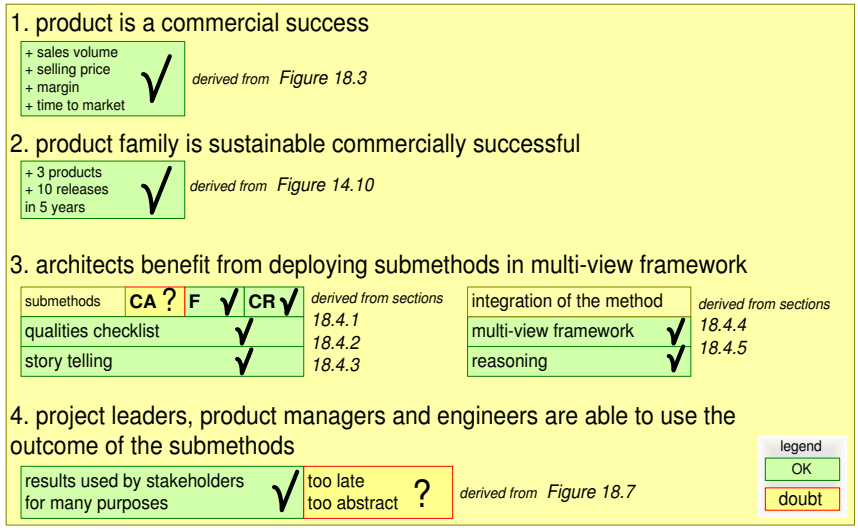


Figure 18.8: The conclusion of the case evaluation

Criterion 2 The product family is sustainable commercially successful (Section 18.3).

Criterion 3 Architects benefit from deploying submethods in a multi-view framework (The submethods, the quality checklist and story telling were respectively evaluated in Subsections 18.4.1, 18.4.2 and 18.4.3). The integration of the method is supported by the multi-view framework and reasoning methods (the integration is evaluated in Subsection 18.4.5. The reasoning method, based on the threads of reasoning, is discussed in Subsection 18.4.4).

Criterion 4 The Product Creation team is used by many stakeholders for many purposes (Section 18.5).

Two evaluation aspects need more evidence or discussion. The submethods in the *Customer Objectives* and *Application* views are insufficiently covered by the case. In the next chapter other supporting evidence for the use of the submethods in these views will be supplied. The perception of some engineers that the results are *too late* or *too abstract* (Section 18.5) will be discussed further in Chapter 20.

Chapter 19

Evaluation from a Wider Context

19.1 Introduction

The architecting method has been illustrated by means of the medical imaging workstation case, and this case was used to evaluate the method. Some aspects of the method could not be evaluated, mostly because these aspects have not been made explicit until after the case period.

In Chapter 5 other evaluation possibilities are indicated, such as *research projects*, *workshops* and *courses*. The significance of this wider context is the potential of performing architecting method research with a greater statistical significance. This chapter does not provide this statistical data. More research, with well defined research protocols, is needed to obtain more robust results. Observations from this context are discussed, however, to show the potential of research in this wider context.

Section 19.2 evaluates the method in the research environment, Section 19.3 in the workshop settings, and Section 19.4 in the course setting. In these sections the suppotive information is indicated by an identification tag:

c n for CAFCR and multi-view information

q n for quality cheklist information

s n for story telling information

i n for iteration information

u n for usability information

In Section 19.5 the findings are summarized.

19.2 Research Environment

For many years research of architecting methods has been performed at Philips Research in the SwA (Software Architectures) group. The research projects in which this group participated have been using some of the discussed architecting (sub)methods consciously.

A short description of the research projects that applied some of the methods is given below:

Family Asset Management How to manage electronic assets, such as movies, pictures, and music? The publication is: [73] is based on this work.

Project infrastructure platform Electronic and software infrastructure (closed circuit TV, security, public access et cetera) to support project organizations in the domains of industrial buildings, banks, railway stations, airport terminals, and motorways.

Heartcare Image and information integration of all cardio-related information from cathlab to personal monitoring used at home or away.In [2] story telling is used in the scenario approach. In [57] story telling, called scenarios in this article, are used to make architectures better future proof.

Platform for portable multi-media Few or single chip electronics and software platform for the creation of mobile multi-media systems (cellphones, PDAs, personal audio, et cetera).

Software productivity for audiovisual systems How to create the software for integrated and connected audio and video systems (TVs, set-top boxes, personal video recorders, DVD recorders, et cetera) in a limited amount of time?

Composable architectures How to create architectures that support composability? This project consolidated and exchanged experiences over a wide range of products: from televisions to cathlabs. The publications [79], [58], [3], and [59] are the first articles and presentations of the BAPO (Business, Architecture, Process and Organization) and CAFCR models. The PhD thesis [48] by Jürgen Müller zooms in on the *Conceptual* and *Realization* views and provides a method to design components that fulfill multiple qualities. In these design views these qualities are called aspects.

In particular the following (sub)methods have been used:

- the decomposition in the 5 CAFCR views
- story telling
- qualities
- iteration over multiple views

Compared to the medical imaging workstation case these research projects put more emphasis on the *Customer objectives* and *Application views* **(c1)**. This results in more focused research projects and less technology push. Researchers in the platform-oriented projects (infrastructure, multi-media, software productivity), for instance, discovered that solutions were being pushed without any clear need at the customer side **(o1)**.

Story (or scenario) telling has been explicitly researched and will be subject of continuing research. The following benefits **(s1)** of story telling were experienced in the *Family Asset management*, *Heartcare*, *portable multi-media* and *software productivity* projects:

- Communication with the less technical stakeholders is improved.
- Exploration discussions are more to the point: less time is lost on too generic discussions

Explicit attention for qualities (based on the qualities checklist in Figure 10.3) also helps to focus the research projects and to find the relevant research issues quicker **(q1)**. Speed of exploration is essential for research projects: identify promising options, and filtering out unattractive options. The speed of exploration is improved by identifying the essential qualities and by identifying the qualities that can be ignored.

Only a very limited improvement in exploration speed has been observed **(i1)**. The highly individual nature of researchers appears to be a bottleneck. Also the diversity and fragmentation of the group of stakeholders, with their individual interests, is a bottleneck. Both bottlenecks hamper the sharing of objectives and the identification of most important qualities. Improvement in exploration speed is certainly possible, but this requires an interaction of architecting with the context of business, processes, and people. New projects at the Embedded Systems Institute, which are set up outside the Philips processes and organization, show promising results. See for instance the Boderc project [23].

No validating or invalidating evidence about the *threads of reasoning* is obtained from the research environment. The *threads of reasoning* did only exist as a vague notion [55].

19.3 Workshops

The architecting methods have also been used to structure many different kinds of workshops. The subjects of these workshops covered areas such as: strategy, roadmapping, project definition exploration of problem and solution domain, cross fertilization, and architecture assessment. The domains that were investigated were quite varied, for example: MR, X-ray, semiconductors, displays, storage, motorway management, and printers.

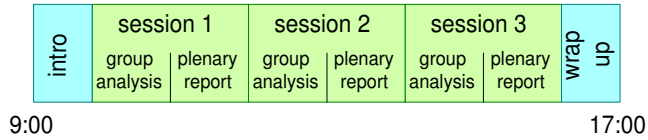


Figure 19.1: Typical workshop program template

Figure 19.1 shows the typical program template of these workshops. Most time is used to stimulate interaction among the participants focused on the subject. This interaction takes place in small teams based on a few predefined questions. The result of these discussions is presented and discussed plenary. A session with the interaction from one team and the plenary presentation typically takes two hours. In a one-day workshop about three successive sessions can be scheduled. The remaining time is needed for introduction and wrap up.

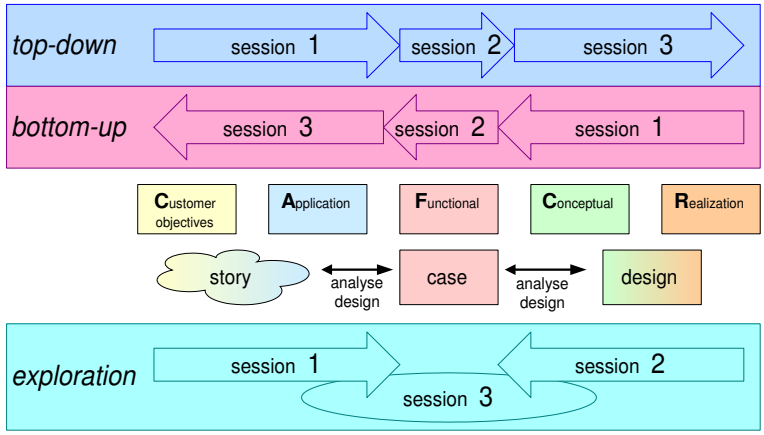


Figure 19.2: Workshop approaches

Figure 19.2 shows several approaches to structure the questions for the three sessions: *top-down*, *bottom-up*, and *exploration*. The basis for all these approaches is the *CAFCR* model **(c2)**, complemented with *story telling* **(s2)**. All approaches have been used with small variations.

The *top-down* approach requires participants that are open and sufficiently customer aware. **What** and **how** questions help the participants to move the investigation from customer towards realization.

In the *bottom-up* approach the link to the customer world is created by repeating **why** questions. The bottom-up approach works well, but should be followed by a top-down question: "Did we start with the *valid* solution?". The improved understanding of the customer often results in adjustments to the original solution. In some cases the participants conclude that the solution is invalid: the solution is not addressing the need of the customer. In that case more appropriate solution directions are generated during the workshop.

The *exploration* approach is more open:

- What is needed?
- What is possible?
- So what are going to create?

The exploration approach is appropriate if sufficient freedom of choice is available; it works less well in very constrained situations.

Also the level of abstraction must be chosen: very generic, for instance identifying key drivers, or very specific via a story. Both generic and specific approaches have been used, as well as combinations of the two. The more generic approaches work well if the participants already have a good shared understanding. If the understanding is limited or not shared specific approaches work better.

The formulation of the questions for the sessions is critical. The questions must be specific to trigger a concrete discussion. The questions must be open to prevent too much bias in the solutions. The CAFCR submethods **(c3)** and the qualities **(q2)** are useful sources of inspiration to articulate the questions. Examples in the *Customer objectives* view are:

- How does the value chain for digital televisions look in 2006?
- What are the key drivers for neuro radiology?

Examples from the *Realization* and *Conceptual* views are:

- What are the most critical system resources for this story? Please quantify.

- What functionality is provided by the Microsoft COM framework? What functionality do we actually use?

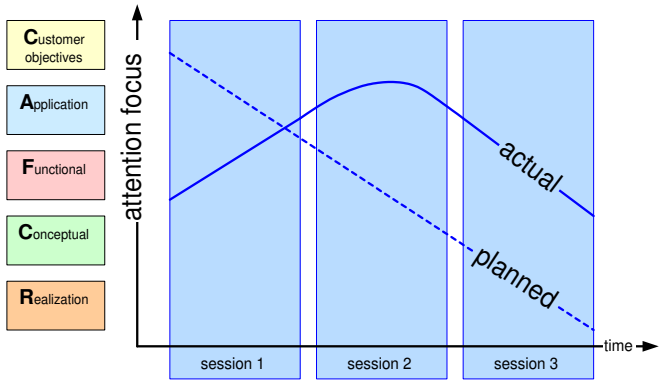


Figure 19.3: Hysteresis due to latency in viewpoint change

The basis of all these approaches is to stimulate the participants to perform a rapid shared iteration. In facilitating more than 30 of these workshops I have observed that the iteration speed in these workshops is limited **(i2)**. Many participants need time to make the context switch. The consequence of this context switch time is that an hysteresis occurs in the goal of the workshop program and the actual execution, as shown in Figure 19.3.

This observation has implications for the usability and efficiency of the architecting method:

- The iteration speed is limited by the capabilities of the architect using them
- The architect must be well aware of the limited iteration speed of his stakeholders. Iterating too quickly in interaction with the stakeholders causes a phase difference between architect and stakeholders. The phase difference has a negative impact on the communication.

System specification and design problems are often caused by the missing links between the *CAFCR* views. Iteration over the *CAFCR* views makes it possible to identify important and critical issues and their relations earlier. Faster iterations bring problems quicker to the surface. In zeroth order[1]: the efficiency of a

[1]If the iteration speed is too high, no practical fact finding and analysis can be applied. A higher order model will show a drop in efficiency for too high iteration speeds. In small (circa 4 people) teams, with a shared background, I have observed useful results in iterations of less than 1 hour **(i3)**.

method is proportional with the iteration speed. The iteration speed is not directly dependent on the method. The speed of iteration is determined by the capabilities of the workshop participants. This again is an area where architecting method and the process and people interfere: the usability of the architecting method depends on the skills and the capabilities of the people and the organization.

19.4 Courses

Some of the submethods are being trained as part of a System Architecting Course. The experiences of teaching this course are described in [54]. As part of the course the participants have to do exercises, some of them using the submethods. The course is mostly focused on the non-technical aspects of system architecting. Five of the course modules, shown in Figure 19.4, have relevant exercises for the evaluation. In the figure is indicated what submethod is used per exercise. The course has been given 20 times between November 1999 and February 2003, with a total amount of participants of about 300.

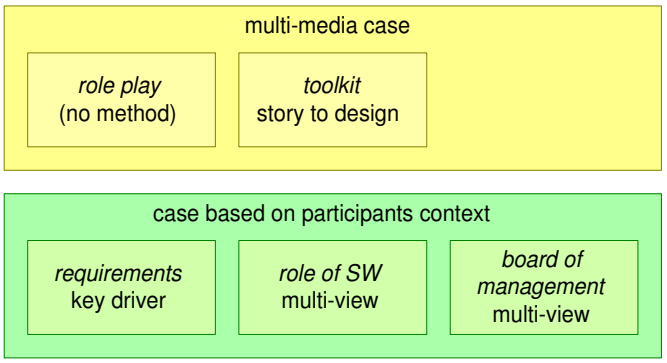


Figure 19.4: Submethods used in course exercises

The *role play*, which is not yet using any method, is relevant because it functions as a kind of zero measurement. The participants play the roles of product manager, project leader, and architect. Together they have to define a new multimedia product, including an indication of business relevance and potential schedule. At the beginning of the course no methods have been provided yet to cope with this kind of problem.

In the *toolkit* exercise newly mixed teams have to use the *story telling* technique to discuss the same product as used in the *role play* exercise. They have to

create a story and to make a start with the analysis and design. Often participants remark that the method would have helped them greatly in the earlier *role play* exercise **(s3)**. The teacher can observe the difference between defining a product without method (the zero measurement) and defining a product with a story telling as method. This order of exercises makes the participants aware of the value of methods. The learning effect increases by experiencing both situations: without and with methods.

The other relevant exercises are all based on the daily context of the participants. The teams are optimized for domain cohesion. Participants are grouped in such a way that they share more or less the same application area and the same type of problems. For example, group names are: *digital video, MR, X-Ray, automotive*, and *optical storage*.

In the *requirements* exercise they have to make a graph, as described in Section 8.2, from key driver to requirements **(c4)**. This is often experienced as an eye-opener: how much more exists than the internal design, how little do we know about the customers!

In the *role of software* the participants have to make a presentation about the software in their system. The explicit recommendation is to do this with multiple diagrams in the Conceptual and Realization view (functional decomposition, layering, flows, size, et cetera) **(c5)**. The presentation should make the intangible software understandable for non-software people. Without this recommendation most engineers tend to explain the software from a single diagram (the class diagram or the layers). Enriching this with other diagrams, such as sizes and other dimensions, helps significantly to make the software more tangible.

The final exercise is a simulated *Board of Management (BoM)* presentation [53], where every team has to give a presentation about an important architectural issue to a management team that is significantly higher up in the hierarchy. They need to deploy a lot of what they learned during the course. To create a successful presentation sufficient customer and business understanding is required (that is the main interest of this higher management team), but it needs to be related to multiple relevant architectural views **(c6)**. The choice of views and submethods is entirely up to the participants.

Many architects struggle in day-to-day life with the perceived lack of understanding of architecting issues by higher management. The *BoM* exercise addresses this problem by improving the presentation content of the (potential) architects. For many participants it is an eye opener to present design issues (Conceptual or Realization views) in relation with the business justification (Customer Objectives, Application, and Functional views).

In the lecture *requirements engineering* for the OOTI-curriculum [74] *CAFCR* and *story telling* are introduced as means to elicit requirements[2]. For most of the postgraduate students customers are far away, they need quite some nudging to take customer needs into consideration. *CAFCR* **(c7)** and *Story telling* **(s4)** clearly help them to think more in customer terms.

19.5 Conclusion

One of the weak spots of the evaluation by means of the medical imaging workstation was the application of submethods in the *Customer objectives* and *Application* views. These submethods have been used much more in all three categories (research, workshops, and courses). This resulted in a more clear project focus and more attention for the customer needs in research projects, compared to previous research projects. Application of these views in workshops improved the attention for the customer needs and the project focus relative to the situation before the workshop. More focus is for most projects of today a big improvement.

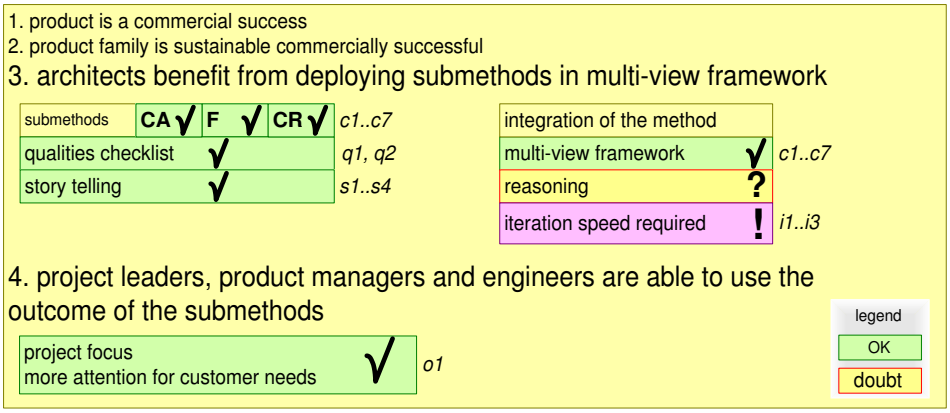


Figure 19.5: Conclusions of the evaluation in a wider context. The tags are defined in the Sections 19.2 to 19.4.

Figure 19.5 shows the conclusions of this evaluation. These conclusions do not address the product and its future, but only the architect and his stakeholders. The use of the *CAFCR multi-view framework* helps to cope with complex product

[2] Also here the hysteresis effect shown in Figure 19.3 is present: in the five days of this lecture and execution of a case it is often difficult to do the iteration more than once; the design analysis is sometimes too superficial, due to the attention on the customer needs.

creation problems. The available *submethods* are successfully used in research environments, workshops, and courses. The qualities from the *quality checklist* helped to bring focus to research projects. No supporting evidence with respect to *threads of reasoning* is obtained from these sources.

A large majority of people lack the skills to iterate very fast over the CAFCR views. They have, however, no problems in following the reasoning when explained. The conclusion is that those who act as architect and deploy this architecting method must have the capability to iterate quickly. In case of an architecting team at least one of the members of the team must have this iterating capability.

Chapter 20

Balancing Genericity and Specificity

20.1 Introduction

The subtitle of this thesis is ”Balancing Genericity and Specificity”. The background of this subtitle is that nearly all development teams of complex systems seem to struggle with this balance.

Section 20.2 discusses this issue in a generic way. Section 20.3 illustrates the the role of this balance in case of the medical imaging workstation. Section 20.4 discusses the interaction of generic and specific elements in the context of the architecting method.

20.2 Core Qualities

The deliverables of the architect are mostly at a high conceptual level. The system specification defines the outline of the system and the system design provides the outline of implementation of the system. Most of the output is rather generic: defining many properties of a system with a minimum set of words and diagrams. The engineers work at all the specific details of the system, which can be millions lines of code or millions of transistors on a chip.

Figure 20.1 shows the two approaches, generic and specific, as complementary approaches in a core quadrant representation, as described by Daniel Offman [60]. The top left quadrant describes the core qualities of a generic systems approach. The top right quadrant describes the pitfalls that occur when the core

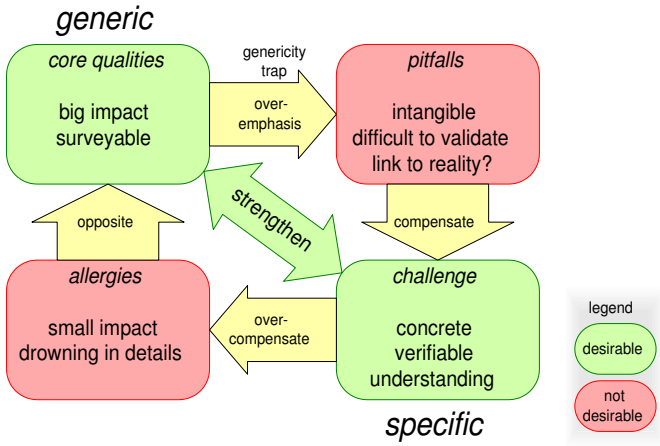


Figure 20.1: Core quadrant representation that shows the complementary nature of generic and specific approaches.

qualities are overemphasized. The bottom right quadrant shows the challenges to prevent the pitfalls to happen. The challenge is to be sufficiently specific. The bottom left quadrant shows the allergies: what happens if too much compensation is applied? These allergies are the opposite of the original core qualities in the top left quadrant.

Generic definitions are very powerful: one sentence may impact thousands or even millions lines of code. The compactness of the output allows all stakeholders to get a good overview of the system and its context.

Working in a too generic way is dangerous: the relation with the real world can be lost (generic motherhood statements). The generic definitions can be rather abstract, thereby making the concepts intangible for many stakeholders. Last but not least, the definitions may have become so generic that their validation is difficult. What is the value of statements that cannot be validated?

Taking a very detailed, highly specific approach has the advantage of being very understandable, due to the closeness with the real implementation and the very concrete nature. Very specific details tend to be easily verifiable.

A disadvantage of a too specific approach is that most individual statements have a very limited impact. Many details have to be specified to cover the entire design. In very specific approaches one can easily get lost and drown in a sea of details.

20.3 Genericity and Specificity in the Case

The entire system specification and design of the medical imaging workstation can be captured in a very limited set of diagrams and tables. Figure 20.2 shows the main diagrams that are needed to understand the design of the medical imaging workstation.

Figure 15.5
SW processes

Figure 15.1
image quality context

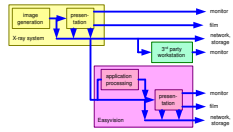
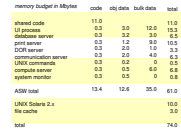


Figure 15.8
memory budget



high level, generic diagrams:
large impact, providing overview

Figure 15.7
construction decomposition

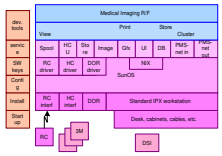
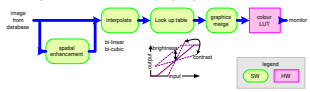


Figure 15.2
processing pipeline



every block, number of word is based on hundreds of specific design details (loc, measurements, images, connections, etc.)

Figure 20.2: Five generic diagrams are shown. Every diagram is based on hundreds of specific design details.

Note that every diagram or table fits on a single A4 or a single slide. The amount of information in every diagram is reduced to the level where the diagram can be explained to stakeholders with a reasonable amount of domain know how in a very limited amount of time. The compactness of the diagram is important to create and maintain overview. The combination of the diagrams creates an integrated understanding of the system design.

Every word, block, or number in every diagram is based on hundreds of details. A block in the construction diagram typically contains 10,000 lines of code, with hundreds of instance variables and hundreds of methods. An arrow in the software process diagram will be used for tens of different connections with hundreds of attributes. Many of these details have been touched by the architect.

However, many more details will never be touched by the architect, as described in Sections 7.2 and 7.5. Submethods such as the *Question Generator*, see Section 9.2.3, help in finding the details to be covered.

The sampling of reality by touching many of the implementation details is the balancing factor in generating the required high-level, compact output. System designs that do not use such fact finding procedures to connect to reality via such fact finding are very risky and do not deserve to be called *system design*.

20.4 Genericity and Specificity in the Architecting Method

The architecting method has many submethods, which invite to be generic. The CAFCR model and the quality checklist are very generic. Every submethod is powerful, and can have a significant impact on the specification and the design. Figure 20.3 positions the architecting method with respect to the level of genericity or specificity, using the scale of abstraction levels from Figure 7.3. The story telling complements the CAFCR submethods and qualities by addressing specific details. The threads of reasoning integrate and balance the generic and specific submethods.

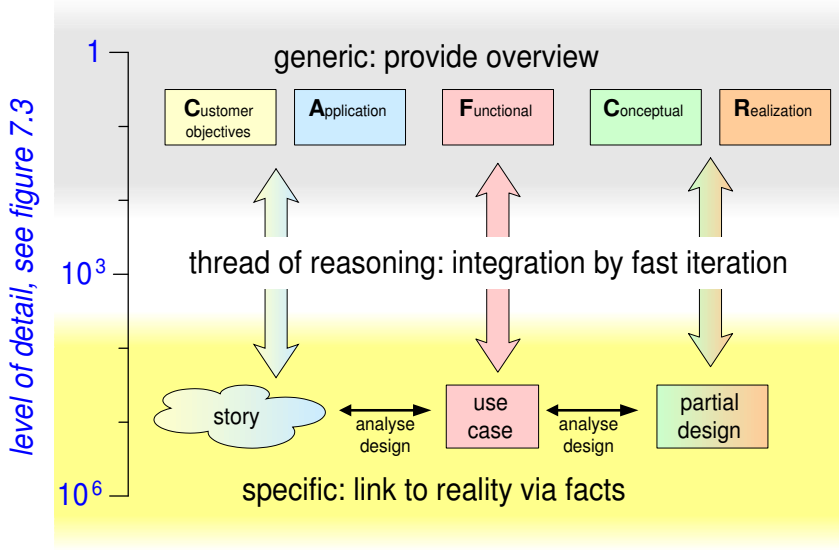


Figure 20.3: The CAFCR views provide generic insight; story telling enables analysis of specific facts.

Story telling is an effective means to make discussions concrete. A good story is **over**specific. On purpose a very narrow, but representative, sample of the target use is described and analyzed. This forces the architect and the designers to look into many specific details.

The threads of reasoning iterate between the generic models and objectives and these details that enable or obstruct the design. The threads of reasoning help to balance genericity and specificity.

Section 7.2 explains the dynamic range of description and implementation facts that needs to be covered by the submethods. The evaluation in Chapter 18 concludes that the method satisfies the needs. The only exception is that the outcome is sometimes too abstract and that the outcome is sometimes too late. The criticism of being too abstract is directly related to the generic nature of the CAFCR submethods. The step from the generic diagrams in Figure 20.2, with hundreds of facts, to the detailed designs of modules and components, with tens of thousands of detailed facts, spans a factor of more than one hundred! Making the generic diagrams more detailed worsens the surveyability and worsens the timely availability, which is the second concern. This tension between the need for genericity for power and understanding, and the need for specificity for the link with reality, is the core problem that architecting methods must tackle.

20.5 Conclusion

The overall architecting methods described in this thesis work successfully in products that span a very large dynamic range of concerns. The criticism of some of the stakeholders clearly identifies a major attention point for the architect when he is deploying the method: the balance between genericity and specificity.

The heuristics for the architect in finding the balance are:

- *Genericity*: overview diagrams fit on a single A4 (20.3).
- *Specificity*: the coverage of detailed facts may vary widely over different parts of the system (20.3).
- *Story telling* facilitates specific discussion, analysis, and design (20.4).
- *Threads of reasoning* integrate generic and specific viewpoints (20.4).

Chapter 21

Reflection on Research Method to Study Architecting Methods

21.1 Introduction

In this chapter we look back at the research method. Where did the research method support the search for successful architecting methods? What aspects of the research methods can be improved?

Section 21.2 discusses the value of the research question. The *hypothesis*, the *criteria*, and the *evaluation* are discussed in Section 21.3. Section 21.4 looks back at the *case description*. The *conclusion* is formulated in Section 21.5.

21.2 Research Question

The explicit formulation of a research question has helped to focus the subject of research. The research objectives and the context have been made explicit. The intention of the research question was to limit the scope to a 'manageable' sized research project.

The human factor is quite dominantly present in the success probability of the architecting method. Figure 21.1 shows the original research question, with a characterization into soft and hard factors. It is immediately clear that many soft factors dominate in the research question. These soft factors can broaden the research scope tremendously. A lot of effort in writing the thesis went into maintaining focus and into balancing hard and soft factors.

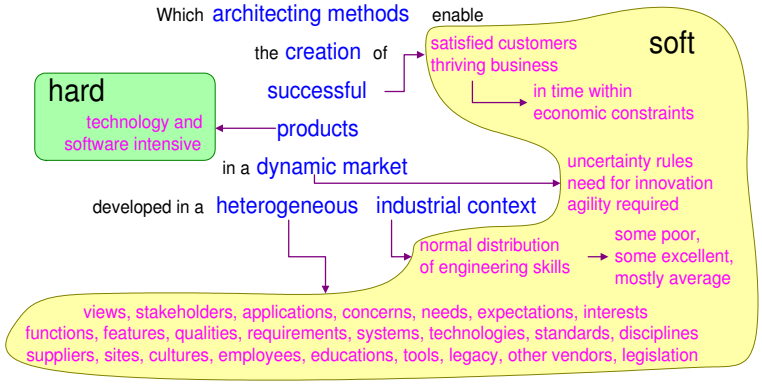


Figure 21.1: The original research question, characterized with hard and soft factors

21.3 Hypothesis, Criteria, and Evaluation

The hypothesis (Section 6.3) extended the research question ((Section 6.2) into a statement that can be validated. The main extension is the addition of **how**. The criteria (Section 6.4) sharpen the hypothesis by adding **who**. The criteria were very valuable in the evaluation, because they focused the evaluation discussion to a limited set of issues. The separation of the criteria for the different stakeholders was essential, because success is measured differently for different stakeholders.

The limitation of this research method is that the hypothesis is only made plausible. The architecting method has been demonstrated successfully in the case and partially in other situations. The hypothesis is not invalidated. In this type of research, with many soft factors, invalidation experiments are difficult: is the hypothesis invalid or did the context not fit in the soft preconditions? Repeated invalidation efforts are needed to increase the plausibility of the method.

21.4 Case Description

The case description is indispensable for this type of research. It illustrates the architecting method much more effectively than any theoretical text can do. The case is also essential to evaluate the hypothesis. The main weakness is that only one case is described. The soft factors are seen as context in this thesis. Soft factors play a dominant role in practice and as shown in Section 21.2. More

case descriptions are needed to separate the method contribution better from the impact of the soft factors. Unfortunately, most cases contain too much sensitive information for the market or the competition. The research can also be extended by including more soft factors in the case description. Describing more soft factors can be privacy sensitive, because it describes behavior of individuals.

Courses and research projects are less competition sensitive. Describing cases from courses and research projects will help to improve the foundation of research methods. Of course, real industrial cases are more supportive than the more indirect cases from research and education.

21.5 Conclusion

The overall research method (research question, hypothesis, criteria, case and evaluation) worked satisfactory, because it helped to articulate the objectives and to focus the research. More case descriptions and more cases describing soft factors will increase the value of this type of research. In Chapter 22 future research directions are discussed.

Chapter 22

The Future of Architecting Research

22.1 Introduction

The development of the system architecting discipline takes place in a dynamic context as shown in Figure 22.1. The natural habitat for system architects is formed by the classical academic disciplines, the standardization bodies, and the communities and conferences. The classical academic disciplines are more mature, but will continue to develop. Many standardization bodies and professional societies are working on system engineering and architecting. Many communities exist where best practices and research results are exchanged in forums, conferences, and many more interactive ways. Sources of inspiration for the development of the system architecting discipline are the management disciplines and the human sciences. Management disciplines are growing, and share the integrating function with the architecting discipline. The human sciences are perceived as *soft*, but as shown in Chapter 21, the architecting discipline will have to cope with many *soft* factors.

The development of the systems architecting discipline involves the creation a frame of reference that helps to relate systems architecting with this context. The associated body of knowledge is discussed in Section 22.2.

Research and consolidation of knowledge is not the only issue to be addressed. The knowledge does get its value if it there are people with the skills to use this knowledge. The translation of the results into an education for system architects is complementary to the knowledge consolidation. Section 22.3 discusses the need

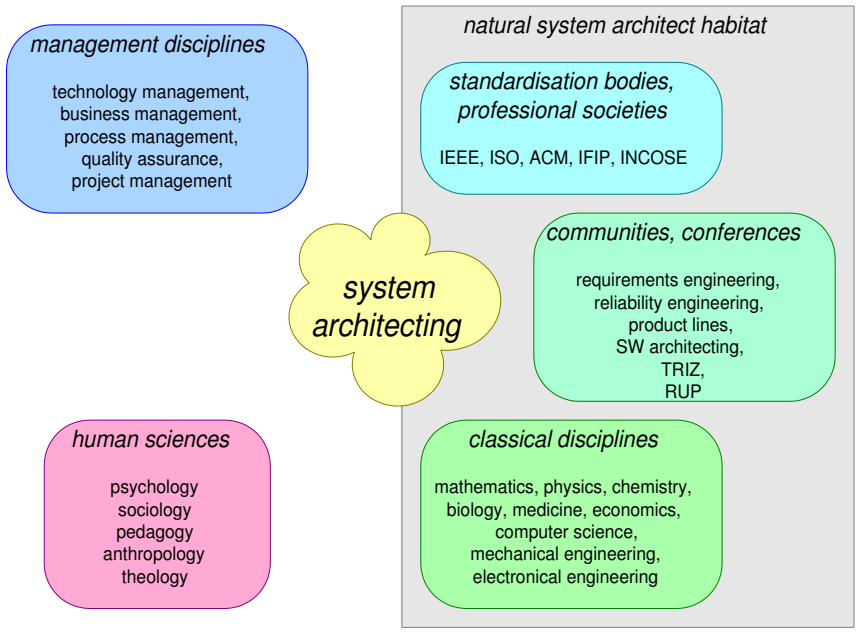


Figure 22.1: The context of architecting

for a systems architecting curriculum.

Section 22.4 summarizes what needs to be done to develop the discipline of system architecting.

22.2 Build up of Body of Knowledge

The body of knowledge to be built consists of different kinds of information. The frame of reference contains reference information, which should help people to position pieces of know-how. For instance:

- classification schemes such as taxonomies or ontologies
- definitions of concepts and terms such as glossaries
- definitions of scope in terms of objectives
- relationships by means of frameworks
- capturing best practices and generic know how in principles and heuristics
- a library of submethods

- a library of case descriptions
- connections with the context
- systematic research of soft factors, for instance by means of questionnaires
- supporting tools

In such a framework we need content, such as a library of case descriptions, and a library of submethods. The methods capture how to approach a system level design problem. A large set of methods is needed, due to the wide variety of problems at system level. Case studies are the carriers of research and consolidation of systems architecting know-how. Very few product developments of industrial size are documented in publicly accessible ways. One of the main hurdles is the confidentiality of the information. It is nevertheless crucial to get a richer set of case descriptions to develop the discipline further. Note that good cases document the product architecture itself, the architecting methods used, and evaluate the use of these methods.

One of the big challenges is to keep up with the growth of functionality, performance, and complexity, as described in Section 5.3. This requires a growth of architecting methods in depth and breadth. More breadth is needed in the capability of handling an even larger dynamic range of abstraction levels. At the same time we have to link the new discipline to the existing sciences, a growth in depth. This link requires a solid research method that facilitates the substantiation of evidence. In general the body of knowledge must be connected to the context shown in Section 22.1: classical disciplines, communities, standardization bodies, management disciplines and the human sciences.

Chapter 19 showed the potential of using courses and workshops to evaluate architecting methods with somewhat more statistical relevance. However, in order to use this source of information we need research methods that work in that environment. The human sciences are much more used to this type of research and can provide inspiration for this type of research methods.

22.3 Curriculum

The typical growth of an architect is used as reference to define desired education steps for architects. The top of Figure 22.2 shows the typical growth of an architect. The growth early in the career is mostly in technical skills. Later the non-technical skills get more attention, in areas such as application, business and process.

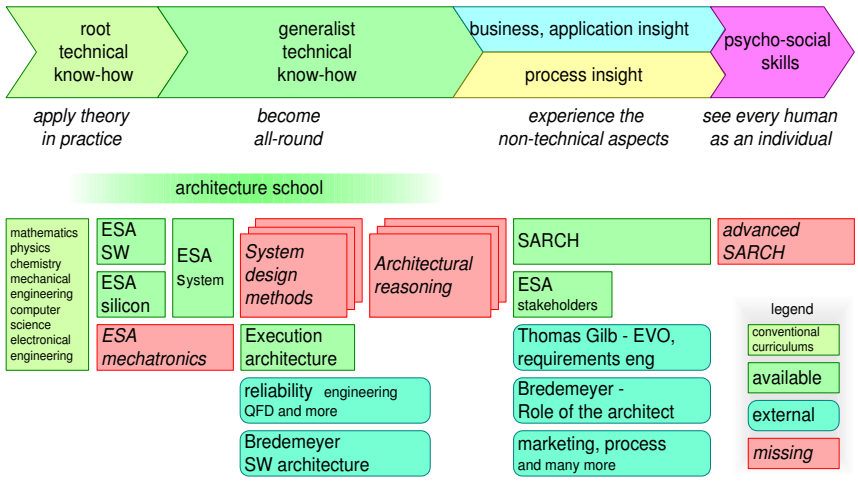


Figure 22.2: Curriculum system architecting

Conventional education is focused on one discipline. This type of education is a good fundament for a potential architect. An architect must have experience in at least one of the technology disciplines, to:

- know and understand detailed technical and engineering problems
- be taken serious as peer for mono-disciplinary engineers

The next step is to broaden the potential architect by providing technical education in other disciplines. At this moment a few courses in this area are available in the ESA (Embedded Systems Architecting) course [24]. This course consists of 4 modules: Software, Silicon, System, and Stakeholders. Software, Silicon, and Systems broaden the technical scope to electronics plus software, illustrated by a number of system technologies such as audio and video. The “harder” disciplines, such as mechatronics, are not yet available in this format.

The real gap in education is in the area of multi disciplinary system design methods. Some communities have created courses in well-defined areas such as reliability engineering (for example FMEA), or in less tangible areas such as Quality Function Deployment (QFD). Many existing design problems, however, are not yet covered by design methods, let alone by education.

The integration of the system level design methods, addressing one or two objectives, into systematic reasoning at system level is the next challenge. Again a lot of method development is needed before education makes sense.

The next maturity step of the architect is supported by borrowing methods and educational material from other disciplines (marketing, process and organization, et cetera).

The most important contribution to the growth of an architect is the practical experience. Working at real problems is crucial. At Philips Research an architecture school is set up, where potential architects work on projects, guided by more experienced engineers. Concurrently they participate in a set of courses, for example the ESA course mentioned above.

22.4 Conclusion

For the development of system architecting as a discipline, we have to :

- develop a framework to position architecting in the context
- create a library of submethods
- develop supporting tools
- build up a library of case descriptions
- research new architecting methods to cope with more breadth and a larger dynamic range
- develop research methods to cope with the soft factors
- create a curriculum to educate potential architects

Chapter 23

Conclusion

We have discussed the industrial context of embedded systems architecting, and the distance between the industrial context and academic research. The challenge to bridge the gap between the industrial and academic worlds is addressed by using a research method based on a hypothesis and a case. The hypothesis is tested by applying the CAFCR method on a Medical Imaging Workstation case. The CAFCR method maps well on this case, although the CA views have been underexposed in the historical context of the case. The use of many different views, submethods, qualities and use cases, as proposed in this thesis, contributed significantly to the success of the Workstation. The beneficial characteristics of the CAFCR method are:

- integral and multi-disciplinary
- goal-oriented
- practical, based on industrial experience
- flexible
- builds on standards
- support for short innovation cycles

This area of research has many connections to both the engineering sciences and the human sciences. One of the main challenges for embedded systems architecting research is to balance the genericity and specificity. Very generic statements are difficult to substantiate and use, but very specific statements might be infeasible due to the sheer amount of required know-how.

Future research work has to be done to substantiate the evaluation from the wider context. In general the area of embedded systems research is young and

lots of research work is required to develop this area, ranging from ontologies and heuristics to extending the library of submethods. Special attention is needed for the many soft factors involved. Based on this research a curriculum has to be developed for the education of system architects.

Part V

Appendices and Bibliography

Acknowledgements

Abbreviations

Bibliography

Summary

Samenvatting

Acknowledgements

Wim Vree coached me in the scientific mores and provided a lot of inspiration. Wim helped in determining the focus in the broad subject of systems architecting, and in structuring the complex material. Wim's enthusiasm contributed a lot to this thesis. Martin Rem has provided lots of feedback on the reasoning and logic of this thesis. As side effect Martin also corrected many language errors, and he provided many hints to improve the readability significantly. Martin's relentless progress and positive attitude were very stimulating.

The other members of the dissertation committee, Peter Kroes, Henk Sips, René Wagenaar, Dieter Hammer, and Pieter Hartel provided helpful feedback after reading the manuscript.

Many managers have encouraged me and facilitated me to write this thesis: Frans Beenker, Rick Harwig, Jaap van der Heijden, Marloes van Lierop, Martin Rem, and Eric van Utteren.

The composable architecture project at Philips Research invented the acronym CAFCR and created the foundation of the CAFCR method. Members of this project were: Pierre America, Marcel Bijsterveld, Peter van den Hamer, Hans Jonkers, Jürgen Müller, Henk Obbink, Rob van Ommering, William van der Sterren, and Jan Gerben Wijnstra.

Many colleagues contributed to this thesis by providing data or feedback. With the danger of forgetting someone I acknowledge the contribution of the following people: Peter Bingley, Robert Deckers, Martien Dijks, Christian Huiban, Eugene Ivanov, Peter Jaspers, Auke Jilderda, Ton Kostelijk, Nico Schellingerhout, Berry van der Wijst, and Rik Willems.

My wife Lia Muller-Charité encouraged me during all the phases preceding the writing of the thesis, as well as during the actual writing. She also helped me by reflecting and discussing the contents itself.

Abbreviations

1471 IEEE standard defining an architecture descriptions

9001 ISO standard defining quality management

9126 ISO standard describing a quality framework

ACR The American College of Radiology

ASML Lithography Company in Veldhoven, the Netherlands

ATAM Architecture Tradeoff Analysis Method by Rick Kazman

BAPO Business Application Process Organization

BoM Board of Management

BoM Bill of Material

CAFCR Customer Objectives, Application, Functional, Conceptual, Realization

C/B Contrast/Brightness

CFO Chief Financial Officer

CIS Cardiology Information System

CMO Chief Marketing Officer

COM Component Object Model, by Microsoft

CoO Cost of Ownership

CPU Central Processing Unit

CT Computer Tomography

CTO Chief Technical Officer

DB DataBase

DICOM Digital Imaging and Communications in Medicine

dll dynamic link library

DOR optical disk

DSP Digital Signal Processor

DVD optical disk succeeding the CD, officially no abbreviation, but some people use it for *Digital Video Disc* or *Digital Versatile Disc*

EMC Electro-Magnetic Compatibility

ESA Embedded Systems Architecting course

ESI Embedded Systems Institute

EVO Evolutionary Project Management method by Thomas Gilb

FDA Food and Drug Administration

FFT Fast Fourier Transform

FMEA Failure Mode Effect Analysis

FRS Functional Requirements Specification

fte Full Time Equivalent, unit of planning indicating a full time available person

GE General Electric

gfx graphics

GHz Giga Hertz

GSM Cell phone standard

GST General Systems Theory

HACCP Hazard Analysis And Critical Control Point

HCU Hardcopy Unit

HD High Definition video

HIPAA Health Insurance Portability and Accountability Act

HIS Hospital Information System

HL7 Health Level 7 standard defining meta information for health care

HQ High Quality audio

HW Hardware

IEEE Institute of Electrical and Electronics Engineers, Inc

INCOSE International Council on Systems Engineering

I/O Input/Output

IQ Image Quality

ISO International Organization for Standardization

IT Information Technology

KOALA a SW component technology used in Philips consumer products

kB kilo Bytes

kloc kilo lines of code

LIS Laboratory Information System

LUT Look Up Table

MB, MByte Mega-Byte

MB, Mbit Mega-bit

MHz Mega Hertz

MLC Material and Labor Cost

MPEG Moving Pictures Experts Group, a compression standard for movies

MPR Multi Planar Reformatting

mrad milliradial

MRI Magnetic Resonance Imaging

MRP Material Resource Planning

NEMA National Electrical Manufacturers Association

nm nanometer, $10^{-}9$ meter

OIT Object Instantiation Tracing

OO Object-Oriented

OS Operating System

OSI Open System Interconnect

PACS Picture Archiving and Communication System

PCP Product Creation Process

PCR Radiography based on Phosphor plate reader

PDA Personal Digital Assistant

PIP Picture In Picture

PMS Philips Medical Systems

PMSnet Philips interoperability protocol, extending the ACR/NEMA or DICOM protocol

ps Unix command to show process statistics

PVR Personal Video Recorder

QFD Quality Function Deployment

RAM Random Access Memory

RC Remote Control

RF Radio Frequency

RIS Radiology Information System

ROI Return On Investment

RUP Rational Unified Process

SAAM A Method for Analyzing the Properties of Software Architectures by Rick Kazman

SARCH Course System Architecting at Center of Technical Training (CTT) of Philips

SD Standard Definition video

SDS System Design Specification]

SE Systems Engineering

SEI Software Engineering Institute

SNR Signal to Noise Ratio

SPC Statistical Process Control

SPS System Performance Specification

SRS System Requirements Specification

SW Software

SwA Software Architectures group at Philips Research

TPD Technical Product Documentation

TPS Test Performance Specification

TRIZ Theory of Inventive Problem Solving

TXT Teletext

UI User Interface

UNIX widely used Operating System

URF Universal Radiography Fluoroscopy

US Ultra Sound

VAP Visual Architecting Process by Bredemeyer

VDE Verband der Elektrotechnik Elektronik Informationstechnik

VDU Video Display Unit

vmstat Unix command to show (virtual) memory statistics

WWHWWW Why What How Where When Whom

WYSIWYG What You See Is What You Get

X Window management system

*x***DAS** Data Acquisition System, the *x* is the version or the type

*x***FEC** Front End Controller, the *x* is the version or the type

ZIFA Zachman Institute for Framework Advancement

Bibliography

[1] Genrich Altshuller. *The Innovation Algorithm; TRIZ, systematic innovation and technical creativity*. Technical Innovation Center, Worcester, MA, 2000. Translated, edited and annotated by Lev Shulyak and Steven Rodman.

[2] Pierre America. Making architectures future-proof using scenarios. `http://www.serc.nl/lac/2003/presentaties/Track1/P.America.pdf`, 2003.

[3] Pierre America, Henk Obbink, Rob van Ommering, and Frank van der Linden. COPAM: A component-oriented platform architecting method family for product family engineering. In Patrick Donohoe, editor, *Proceedings of the First Software Product Line Conference*, August 2000.

[4] Audrey Apfel. BVIT: Frameworks and methodologies that work. `http://www3.gartner.com/resources/113500/113516/113516.pdf`, 2003.

[5] Apple Computer, Inc. The Objective-C Programming Language. `http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/index.html`, 2003.

[6] Architecture Working Group (AWG). *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*. The Institute of Electrical and Electronics Engineers, Inc., 2000.

[7] Arthur D Little Global Management Consultants. Technology management study, 1998. confidential report at ASML. Complete derivation from 5 customer key driver to about 250 critical waferstepper technologies.

[8] J. E. Beasley. OR-notes. `http://mscmga.ms.ic.ac.uk/jeb/or/basicor.html`.

[9] Per Bjuréus and Axel Jantsch. MASCOT: A specification and cosimulation method integrating data and control flow. http://jamaica.ee.pitt.edu/Archives/ProceedingArchives/Date/Date2000/papers/2000/date00/pdffiles/03a_2.pdf, 2000.

[10] Dana Bredemeyer. Definitions of software architecture. http://www.bredemeyer.com/definiti.htm, 2002. large collection of definitions of software architecture.

[11] Dana Bredemeyer and Ruth Malan. Resources for software architects. http://www.bredemeyer.com/, 1999.

[12] Dana Bredemeyer and Ruth Malan. Role of the software architect. http://www.bredemeyer.com/role.pdf, 1999.

[13] Dana Bredemeyer and Ruth Malan. The visual architecting process. http://www.bredemeyer.com/pdf_files/WhitePapers/VisualArchitectingProcess.PDF, 2003.

[14] Florian Cajori. *A history of Mathematical Notations*. The Open Court Publishing Company, 1928.

[15] Samidh Chakrabarti and Aaron Strauss. Carnival booth: An algorithm for defeating the computer-assisted passenger screening system. http://swissnet.ai.mit.edu/6805/student-papers/spring02-papers/caps.htm, 2002. Shows that security systems based on secret designs are more vulnerable and less secure.

[16] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.

[17] Hay Management Consultants. Technology management cycle. Hay Managament Consultants showed me this model in 1997/1998, taken from an article by a Japanese author. The original title of the Japanese article is unknown.

[18] Gerardo Daalderop, Ann Ouvry, Luc Koch, Peter Jaspers, Jürgen Müller, and Gerrit Muller. PACS assessment final report, version 1.0. confidential internal report XLB050-96037, September 1996.

[19] Paul de Witte. CTT course SARCH. http://www.extra.research.philips.com/ctt/ctt_it/sarch.htm, 1999.

[20] Jean-Marc DeBaud and Klaus Schmid. A systematic approach to derive the scope of software product lines. In *21st international Conference on Software Engineering; Preparing for the Software Century*, pages 34–47. ICSE, 1999.

[21] J. C. DeFoe (Editor). An identification of pragmatic principles. `http://www.incose.org/workgrps/practice/pragprin.html`, 1999.

[22] Remco M. Dijkman, Luís Ferreira Pires, and Stef M.M. Joosten. Calculating with concepts: a technique for the development of business process support. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, *Lecture Notes in Informatics*, volume 7, pages 87–98. GI-edition, 2001. `http://www.google.com/url?sa=U&start=3&q=http://www.ub.utwente.nl/webdocs/ctit/1/00000068.pdf&e=7764` Proceedings of the UML 2001 Workshop on Practical UML-Based Rigorous Development Methods.

[23] Embedded Systems Institute. Boderc project. `https://www.embeddedsystems.nl/PRO1/general/next.asp?subrubriekid=2`, 2003.

[24] Embedded Systems Institute. Course on embedded systems architecting. `https://www.embeddedsystems.nl/PRO1/general/show_document_general.asp?documentid=676`, 2003.

[25] EventHelix.com. Publish-subscribe design patterns. `http://www.eventhelix.com/RealtimeMantra/Patterns/publish_subscribe_patterns.htm`, 2000.

[26] Thomas Gilb. Competitive engineering. `http://www.pimsl.com/TomGilb/Competitive_Engineering_M.pdf`, 1999.

[27] H Gomaa. *Software Design Methods for Real-time Systems*. Addison-Wesley, 1993.

[28] Robert J. Graham and Randall L. Englund. *Creating an Environment for Successful Projects; The Quest to Manage Project Management*. Jossey-Bass Publishers, San Fransisco, CA, 1997.

[29] Volker Haarslev. A fully formalized theory for describing visual notations. In *Proceedings, International Workshop on Theory of Visual Languages, Gubbio, Italy*, 1996.

[30] John L. Hennessy, David A. Patterson, and David Goldberg. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[31] F. Heylighen and C. Joslyn. What is systems theory? `http://pespmc1.vub.ac.be/SYSTHEOR.html`, 1992. *Principia Cybernetica Web* (Principia Cybernetica, Brussels).

[32] Derek K. Hitchins. Putting systems to work. `http://www.hitchins.co.uk/`, 1992. Originally published by John Wiley and Sons, Chichester, UK, in 1992.

[33] Christine Hofmeinster, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley, 2000.

[34] Laurence Holt. *Goal-oriented design*. unknown, 2004 (planned). A preview was presented at the Gilb Systecture event 2003, London, June 2003.

[35] Robert Hunt. The origins of proof iv: The philosophy of proof. `http://plus.maths.org/issue10/features/proof4/`, 2000.

[36] INCOSE. International council on systems engineering. `http://www.incose.org/toc.html`, 1999. INCOSE publishes many interesting articles about systems engineering.

[37] Carnegie Mellon Software Engineering Institute. How do you define software architecture? `http://www.sei.cmu.edu/architecture/definitions.html`, 2002. large collection of definitions of software architecture.

[38] ISO/IEC. ISO 9126: The standard of reference. `http://www.cse.dcu.ie/essiscope/sm2/9126ref.html`, 1991.

[39] Hans Jonkers. Interface-centric architecture descriptions. In *WICSA 2001, Amsterdam*, 2001.

[40] R. Kazman, M. Klein, and P. Clements. ATAM: Method for architecture evaluation. `citeseer.nj.nec.com/kazman00atam.html`, 2000.

[41] Rick Kazman, Leonard J. Bass, Mike Webb, and Gregory D. Abowd. SAAM: A method for analyzing the properties of software architectures. In *International Conference on Software Engineering*, pages 81–90. ICSE, 1994.

[42] Kurt Keutzer and Richard Newton. Productivity gap. http://lark.vmei.acad.bg/asic/lectures/intro/slide1-5.htm, 2000.

[43] Philippe B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, pages 42–50, November 1995.

[44] Ben Lieberman. The art of modeling; part i: Constructing an analytical framework. http://www.therationaledge.com/content/aug_03/f_modeling_bl.jsp, 2003.

[45] Charles C. Mann. Homeland insecurity. *The Atlantic Monthly*, pages 81–102, September 2002. Volume 290, No. 2T; Very nice interview with Bruce Schneier about security and the human factor.

[46] James N. Martin. *Systems Engineering Guidebook*. CRC Press, Boca Raton, Florida, 1996.

[47] Carver Mead and Lynn Conway, editors. *Introduction to VLSI systems*. Addison-Wesley, 1980. Chapter 7: System Timing, by Charles L. Seitz.

[48] Jürgen K. Müller. *The Building Block Method: Component-Based Architectural Design for Large Software-Intensive Product Families*. Universiteit van Amsterdam, 2003. Ph.D. thesis.

[49] James Mohr. The linux tutorial; run-levels. http://www.linux-tutorial.info/cgi-bin/display.pl?65&99980&0&3, 1997.

[50] Gerrit Muller. Technology improvement plan. confidential internal report, June 1994.

[51] Gerrit Muller. CTT course SARCH. http://www.extra.research.philips.com/natlab/sysarch/SARCHcoursePaper.pdf, 1999.

[52] Gerrit Muller. Case study: Medical imaging; from toolbox to product to platform. http://www.extra.research.philips.com/natlab/sysarch/MedicalImagingPaper.pdf, 2000.

[53] Gerrit Muller. How to present architecture issues to higher management. http://www.extra.research.philips.com/natlab/sysarch/ArchitectManagementInteractionPaper.pdf, 2003.

[54] Gerrit Muller. Experiences of teaching systems architecting. To be published INCOSE 2004 in Toulouse, 2004.

[55] Gerrit Muller, Jürgen Müller, and Jan Gerben Wijnstra. Multi-view architecting. `http://www.extra.research.philips.com/natlab/sysarch/IntegratingCAFCRPaper.pdf`, 2001.

[56] Peter G. Neumann. Homepage peter g. neumann a.o. about safety, security and reliability. `http://www.csl.sri.com/users/neumann/`.

[57] Henk Obbink. Scenario-based architecting: Towards architecting the future. `http://www.serc.nl/lac/2003/presentaties/Track1/H.Obbink.pdf`, 2003.

[58] Henk Obbink, Jürgen Müller, Pierre America, and Rob van Ommering. COPA; a component-oriented platform architecting method for families of software-intensive electronic products. `http://www.extra.research.philips.com/SAE/COPA/COPA_Tutorial.pdf`, 2000.

[59] Henk Obbink, Rob van Ommering, Jan Gerben Wijnstra, and Pierre America. Component oriented platform architecting for software intensive product families. In Mehmet Aksit, editor, *Proceedings of Software Architectures and Component Technology*. Kluwer Enschede, January 2000.

[60] Daniel Offman. Bezieling en kwaliteit in organisaties.

[61] David L. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, pages 128–138, March 1979. This article can also be found in "Software Fundamentals, Collected Papers by David Parnas", Addison-Wesley.

[62] D.L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, SE-12., No. 2:251–257, February 1986.

[63] Karen Peterson, David Hope-Ross, Andrew White, and Marc Halpern. Deriving competitive advantage from product value chains. `http://www3.gartner.com/Init`, 2002.

[64] Michael J. Pidwirny. Fundamentals of physical geography; chapter 3a scientific method. http://www.geog.ouc.bc.ca/physgeog/contents/3a.html, 1996.

[65] Colin Potts. Software-engingeering research revisited. *IEEE Software*, Vol. 10, No. 5:19–28, September/October 1993.

[66] William H. Press, William T. Vetterling, Teulosky Saul A., and Brian P. Flannery. *Numerical Recipes in C; The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 1992. Simulated annealing methods page 444 and further.

[67] QFD Institute. QFD institute. http://www.qfdi.org/, 2000.

[68] Eberhardt Rechtin and Mark W. Maier. *The Art of Systems Architecting*. CRC Press, Boca Raton, Florida, 1997.

[69] Martin Rem. Trends in embedded systems. http://www.hightechconnections.org/presentations/martinrem.pdf, 2004.

[70] Oliver W. Sacks. *The Man Who Mistook His Wife for a Hat: And Other Clinical Tales*. Touchstone Books, 1985. Oliver Sacks has published a rich collection of case descriptions, much more than the descriptions in this book. He explains in this book the value of case descriptions. Interesting is that although dr. Sacks is a neurologist, the case descriptions are much richer and contain many psycho social observations as well.

[71] Carnegie Mellon Software Engineering Institute SEI. Software engineering management practices. http://www.sei.cmu.edu/managing/managing.html, 2000.

[72] Carnegie Mellon Software Engineering Institute SEI. Engineering practices. http://www.sei.cmu.edu/engineering/engineering.html, 2002.

[73] Alexander Sinitsyn. A synchronization framework for personal mobile servers. PerWare '04: Middleware Support for Pervasive Computing; Workshop (at 2nd Conference on Pervasive Computing), March 2004.

[74] Stan Ackermans Institute. Ooti: Post-masters program in software technology. http://wwwooti.win.tue.nl/, 2004.

[75] The Medical HACCP Alliance. Hazard analysis and critical control point. http://medicalhaccp.ag.vt.edu/, 1998.

[76] Henk Tuten. Popper and philosophy of science. http://huizen.daxis.nl/~henkt/popper-scientific-philosophy.html, 1999.

[77] Henk Tuten. Thomas kuhn: definition paradigm (shift). http://huizen.daxis.nl/~henkt/kuhn.html, 1999.

[78] Rob van Ommering. Building product populations with software components. In *ICSE 2002*, 2002.

[79] Jan Gerben Wijnstra. Critical factors for a successful platform-based product family approach. In *Proceedings of the Third Software Product Line Conference*, August 2002.

[80] John Zachman. The zachman framework for enterprise architecture. http://www.zifa.com/, 1987.

[81] H Zimmermann. OSI reference model - the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, COM-28, No. 4, April 1980.

Summary

This thesis describes an architecting method for embedded systems. Examples of embedded systems are cell phones, televisions, MRI scanners and wafersteppers. Embedded systems are systems where the computer and the accompanying software are built-in. Moreover, embedded systems are full of other technologies that are needed to perform the function in the physical world. Examples are transmission and reception technologies, display technology, optics and all kind of mechanics to position.

The core of the described method is the use of multiple viewpoints on the system and the application of the system. The CAFCR model provides 5 viewpoints: Customer Objectives (**what** does the customer want to achieve), Application (**how** does the customer realize these objectives), Functional (**what** must the system do), Conceptual and Realization(**how** will the system be implemented). The Conceptual view contains the reusable concepts of the design, while the realization contains all the technical implementation details. These five views are related by the quality attributes, such as safety, performance and functionality.

For every CAFCR viewpoint multiple submethods are shown. A system architect makes a choice from the presented methods. Every application domain has specific characteristics that has to be taken into account by the system architect. The choice of the methods to be used depends on the application domain.

One of the most difficult choices for the system architect is the level of abstraction. *Story telling* is recommended as a method to have concrete discussions. Story telling should precede the more generic specification phase.

The system architect must maintain overview in the multitude of viewpoints, objectives, stakeholders, concerns, technologies and design choices. Moreover, the system architect has to make choices that fit in the bottomline goals of the system and the business. It is recommended to iterate fast over all viewpoints and to generate a graph of relationships between objectives and technology choices.

The method is applied in retrospect on a Medical Imaging Workstation. The

casus shows the importance of using multiple viewpoints to analyze multiple qualities.

Architecting of Embedded Systems is a young discipline. There is not much literature available and the education towards system architect is rather immature still. In this thesis it is indicated how to start research in this young discipline. it is also indicated what more has to be done to develop this discipline.

Samenvatting

Dit proefschrift beschrijft een ontwerpmethode voor embedded systemen. Voorbeelden van embedded systemen zijn GSM telefoons, televisies, MRI scanners en wafersteppers. Embedded systemen zijn systemen waar de computer met de bijbehorende software is ingebouwd. Bovendien zijn embedded systemen rijk aan andere technologieën die nodig zijn om het systeem zijn functie in de physieke wereld te laten vervullen. Voorbeelden zijn zend- en ontvangsttechnologie, beeldschermtechnologie, optiek en allerlei mechanieken om te positioneren.

De kern van de omschreven methode is het gebruik van vele verschillende gezichtspunten op het systeem en de toepassing van het systeem. Het CAFCR model geeft 5 gezichtspunten: Doelstellingen van de klant (**wat** wil de klant bereiken), toepassing (**hoe** realiseert de klant deze doelstellingen), functioneel (**wat** moet het systeem doen), conceptueel en realisatie (**hoe** gaat het systeem gebouwd worden). Het conceptuele gezichtspunt bevat de meer herbruikbare concepten van het ontwerp, terwijl het realisatie-gezichtspunt alle technische implementatie details bevat. Deze vijf gezichtspunten kunnen aan elkaar gerelateerd worden door te kijken naar de gewenste kwaliteitseigenschappen, zoals veiligheid, snelheid en functionaliteit.

Voor de CAFCR gezichtpunten worden meerdere deelmethodes getoond. Een systeemontwerper kan een keuze maken uit de geboden methodes. Ieder toepassingsgebied heeft specifieke eigenschappen waar de systeemontwerper rekening mee moet houden. De keuze van de te gebruiken deelmethodes wordt bepaald door het toepassingsgebied.

Een van de moeilijkste keuzes voor de systeemontwerper is het niveau van abstractie. Het vertellen van verhalen wordt aanbevolen als methode om discussies voldoende concreet te krijgen, voordat de meer abstracte specificaties worden opgesteld.

In de veelheid aan gezichtspunten, doelstellingen, belanghebbenden, zorgen, technologieën en ontwerpkeuzes moet de systeemontwerper het overzicht bewaren.

Bovendien moet de systeemontwerper daarbij keuzes maken die passen binnen de uiteindelijke doelstellingen van het systeem en van het bedrijf. Het wordt aanbevolen om hiertoe snel te itereren over de gezichtspunten en om een samenhangende graaf te creëren. De graaf legt de relaties tussen doelstellingen en technologiekeuzes.

De gehele methode wordt onderbouwd door de methode retrospectief toe te passen op een medisch beeldverwerkingsstation. In deze casus wordt getoond hoe belangrijk het is om vanuit meerdere gezichtspunten naar meerdere kwaliteitseigenschappen te kijken.

Het ontwerpen van ingewikkelde embedded systemen is een heel jong vakgebied. Er is weinig literatuur over beschikbaar en de opleiding tot systeemontwerper staat ook nog in de kinderschoenen. In dit proefschrift wordt een aanzet gegeven hoe onderzoek te doen in dit jonge vakgebied. Ook wordt kort aangegeven wat er moet gebeuren om het vakgebied verder te ontwikkelen.

History

Version: 2.9, date: April 21, 2004 changed by: Gerrit Muller

- fine tuning of layout
- updated the acknowledgements

Version: 2.8, date: April 19, 2004 changed by: Gerrit Muller

- added sentence about success of the method in the case

Version: 2.7, date: April 14, 2004 changed by: Gerrit Muller

- changed layout; chapters start at right hand pages only

Version: 2.6, date: April 7, 2004 changed by: Gerrit Muller

- corrected bibliography entries
- added Chapter Conclusion to part IV
- created part V for appendices and bibliography
- added table of content for each part
- some chapter names changed

Version: 2.5, date: April 6, 2004 changed by: Gerrit Muller

- moved Figure Structure of thesis from Criteria chapter to Introduction.
- changed status to finished

Version: 2.4, date: April 1, 2004 changed by: Gerrit Muller

- added chapter references to Introduction.
- textual changes to Introduction
- changed title of Part I into “Introduction to CAFCR and Threads of Reasoning”

Version: 2.3, date: March 22, 2004 changed by: Gerrit Muller

- added stubs for Preface, Summary and Samenvatting
- removed abstract

Version: 2.2, date: March 16, 2004 changed by: Gerrit Muller

- rewrite of the introduction
- changed status to concept

Version: 2.1, date: February 27, 2004 changed by: Gerrit Muller

- added “conclusion” or “summary” sections to chapters in Part II and Part III

Version: 2.0, date: January 21, 2004 changed by: Gerrit Muller

- removed list of figures
- changed formfactor to B5
- changed frontmatter and backmatter structure

Version: 1.2, date: January 19, 2004 changed by: Gerrit Muller

- updated acknowledgements

Version: 1.1, date: December 5, 2003 changed by: Gerrit Muller

- updated abstract
- changed title in ”CAFCR: A Multi-view Method for Embedded Systems Architecting”

Version: 1.0, date: November 21, 2003 changed by: Gerrit Muller

- textual changes in introduction
- changed status in ”draft”

Version: 0.4, date: October 11, 2003 changed by: Gerrit Muller

- extended the list of abbreviations.

Version: 0.3, date: October 2, 2003 changed by: Gerrit Muller

- moved the Chapter ”Criterions for architecting methods” to the end of Part I.

Version: 0.2, date: September 29, 2003 changed by: Gerrit Muller

- moved the introduction of the case more up front.
- moved chronological case description to the beginning of Part III.

Version: 0.1, date: September 17, 2003 changed by: Gerrit Muller

- Added Appendix "Abbreviations"

Version: 0, date: July 28, 2003 changed by: Gerrit Muller

- Thesis created by refactoring the book into a scientific oriented thesis and a educational oriented book